APPENDIX XI.
RELATED CASES

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Ex parte BART ALAN MELTZER, TERRY ALLEN, MATTHEW DANIEL FUCHS, ROBERT JOHN GLUSHKO, and MURRAY MALONEY

Appeal No. 2006-1639
Application No. 09/173,858

ON BRIEF

Before KRASS, JERRY SMITH, and BLANKENSHIP, Administrative Patent Judges.

JERRY SMITH, Administrative Patent Judge.

DECISION ON APPEAL

This is a decision on the appeal under 35 U.S.C. § 134 from the examiner's rejection of claims 1-16 and 61-72.

The disclosed invention pertains to documents for commerce in trading partner networks and interface definitions based on the documents.

Representative claims 1 and 61 are reproduced as follows:

1. An interface for transactions among nodes in a network including a plurality of nodes which execute processes involved in the transactions, the interface being stored in a computer readable medium, comprising:

- a machine readable specification of an interface to transaction processes stored in memory accessible by at least one node in the network, including interpretation information providing a definition of an input document, and a definition of an output document, the definitions of the input and output documents comprising respective descriptions of sets of storage units and logical structures for the sets of storage units.

61. A method for programming a commercial transaction in a network, comprising:

- defining a machine readable definition of an input document for a node in the network including resources to execute a process in the transaction, and a machine readable definition of an output document for the node, the definitions of the input and output documents comprising respective descriptions of sets of storage units and logical structures for the sets of storage units; and

- providing interpretation information for the logical structures to the node.

The examiner relies on the following references:

- McKendrick, Joseph, (McKendrick) "Banks begin to play with XML", Bank Technology News, New York, Sep. 1998, Vol. 11, Issue 9, pg. 6, 2 pgs.
- W3C, (W3C) "Extensible Markup Language (XML) 1.0", Feb. 10, 1998, pages 1-37.

The following rejection is on appeal before us:

1. Claims 1-16 and 61-72 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over the teachings of McKendrick in view of W3C [answer, page 3].

Rather than repeat the arguments of appellants or the examiner, we make reference to the briefs and the answer for the respective details thereof.

We have carefully considered the subject matter on appeal, the rejection advanced by the examiner and the evidence of obviousness relied upon by the examiner as support for the rejection. We have, likewise,

reviewed and taken into consideration, in reaching our decision, the appellants' arguments set forth in the briefs along with the examiner's rationale in support of the rejection and arguments in rebuttal set forth in the examiner's answer. Only those arguments actually made by appellants have been considered in this decision. Arguments which appellants could have made but chose not to make in the briefs have not been considered and are deemed to be waived. See 37 C.F.R. § 41.37(c)(1)(vii)(2004). See also In re Watts, 354 F.3d 1362, 1368, 69 USPQ2d 1453, 1458 (Fed. Cir. 2004).

It is our view, after consideration of the record before us, that the evidence relied upon by the examiner does support the examiner's rejection of claims 1-16 and 61-72. Accordingly, we affirm.

37 C.F.R. § 1.131 Declaration

I. We consider first the Rule 131 declaration on its merits. See MPEP §715.08. Title 37 C.F.R. § 1.131 provides in pertinent part:

§ 1.131 Affidavit or declaration of prior invention.

(a) When any claim of an application or a patent under reexamination is rejected, the inventor of the subject matter of the rejected claim, the owner of the patent under reexamination, or the party qualified under §§ **1.42**, **1.43**, or **1.47**, may submit an appropriate oath or declaration to establish invention of the subject matter of the rejected claim prior to the effective date of the reference or activity on which the rejection is based. The effective date of a U.S. patent, U.S. patent application publication, or international application publication under PCT Article **21(2)** is the earlier of its publication date or date that it is effective as a reference under 35 U.S.C. **102(e)**. Prior

invention may not be established under this section in any country other than the United States, a NAFTA country, or a WTO member country. Prior invention may not be established under this section before December 8, 1993, in a NAFTA country other than the United States, or before January 1, 1996, in a WTO member country other than a NAFTA country. Prior invention may not be established under this section if either:

(1) The rejection is based upon a U.S. patent or U.S. patent application publication of a pending or patented application to another or others which claims the same patentable invention as defined in § **41.203**(a) of this title, in which case an applicant may suggest an interference pursuant to § **41.202**(a) of this title; or

(2) The rejection is based upon a statutory bar.

(b) The showing of facts shall be such, in character and weight, as to establish reduction to practice prior to the effective date of the reference, or conception of the invention prior to the effective date of the reference coupled with due diligence from prior to said date to a subsequent reduction to practice or to the filing of the application. Original exhibits of drawings or records, or photocopies thereof, must accompany and form part of the affidavit or declaration or their absence must be satisfactorily explained.

We begin by noting that the filing date of the instant application is Oct. 16, 1998. We note that appellants bear the burden of production to present evidence of an asserted date of invention prior to the instant application filing date. We note that McKendrick is considered as a printed publication published in the United States that qualifies as prior art under 35 U.S.C. § 102(a), and therefore may be antedated by a declaration that complies with the requirements of 37 C.F.R. § 1.131(b). We further note that the critical date to overcome is the Sept. 1998 publication date of the McKendrick reference [McKendrick, see pub. date shown on page 1]. Thus, appellants have to show either: (1) a reduction to practice, or (2) conception of the invention plus diligence to actual or constructive reduction to practice

before the critical date. 37 C.F.R. § 1.131(b). We note that appellants assert possession of the invention before the critical date by presenting evidence purportedly demonstrating an actual reduction to practice prior to March 11, 1998 [see Rule 131 declaration received Jan. 31, 2005, including Exhibit A].

In the Rule 131 declaration, appellants make the following assertions in reference to a "memorandum" that is presented as Exhibit A [Rule 131 declaration, pages 1 and 2]:

- Prior to March 11, 1998, we had implemented a registry for trading partners. The registry was used in a method, also implemented prior to March 11, 1998, in a form sufficient to demonstrate that the method would work for its intended purpose, for establishing transactions among trading partners in a network, comprising: maintaining a registry of machine-readable specifications specifying business services offered by trading partners, the machine-readable specifications including at least one of definitions of, and references to definitions of, services offered and at least one of definitions of, and references to destinations of, documents to be exchanged with such services by trading partners; and providing, in response to a request, one or more of the machine-readable specifications from said registry is via a communication network to a requesting node.
- Attached hereto as Exhibit A is an excerpt of a memorandum, which I am informed, or know from personal knowledge, was written by co-inventor Glushko, prior to March 11, 1998. Exhibit A includes the statement "In particular, the eCo server has now subsumed the registry and query services that had been envisioned as part of the Taxonomy of Everything in our proposal." This comment establishes that the registry and supporting services had been implemented at the time the memorandum was written.
- The implementation of the method described above occurred within the United States prior to March 11, 1998.
- Work on the system including the registry continued without interruption from at least as early as March 11, 1998, through the filing date of the parent U.S. patent application No. 09/173,854 on October 16, 1998.

It is well established that proof of actual reduction to practice requires demonstration that the embodiment relied upon as evidence of priority actually worked for its intended purpose. Newkirk v. Lulejian, 825 F.2d

1581, 1582, 3 USPQ2d 1793, 1794 (Fed. Cir. 1987) citing Wiesner v. Weigert, 666 F.2d 582, 588, 212 USPQ 721, 726 (CCPA 1981) ("an invention is not reduced to practice until its practicability or utility is demonstrated pursuant to its intended purpose"); Chandler v. Mock, 150 F.2d 563, 565, 66 USPQ 209, 211 (CCPA 1945) ("reduction to practice of a complex mechanical device ... requires that the device was subjected to a test under actual working conditions which demonstrated not that the device might work, but that it actually did"). Affidavits and declarations fail in their purpose when they recite conclusions with few facts to buttress the conclusions. In re Brandstadter, 484 F.2d 1395, 1404, 179 USPQ 286, 292 (CCPA 1973).

In the instant case, we have carefully considered the Rule 131 declaration and evidence (Exhibit A) and we do not agree that the evidence shows an actual reduction to practice of the instant claimed invention prior to March 11, 1998. We note that it is appellants' burden to clearly explain how the proffered evidence (i.e., Exhibit A) shows completion of the invention. See In re Borkowski, 505 F.2d 713, 719, 184 USPQ 29, 33-34 (CCPA 1974) ("It was appellants' burden to explain the content of these notebook pages as proof of acts amounting to reduction to practice. That was not done."). Vague and general statements in a declaration with respect to what the exhibits show along with the assertion that the exhibits describe a reduction to practice are insufficient. See id. at 718, 184 USPQ at

33 (vague and general statements amount to mere pleading, unsupported by proof or showing of facts). After careful consideration of the evidence before us, we conclude that appellants have failed to provide a factual showing that the embodiment relied upon actually worked for its intended purpose as required to demonstrate an actual reduction to practice.

Furthermore, we find that appellants have failed to completely read the language of the instant claims on the proffered Exhibit A evidence provided in the declaration. In particular, we note that appellants have failed to point to specific portions of Exhibit A that demonstrate actual reduction to practice of the instant claimed "interpretation information providing a definition of an input document, and a definition of an output document, the definitions of the input and output documents comprising respective descriptions of sets of storage units and logical structures for the sets of storage units" [claim 1, emphasis added]. While we recognize that XML (i.e., Extensible Markup Language) may provide Document Type Definitions (see e.g., W3C, p. 9, §2.8), the mere use of XML does not disclose input and output documents, per se. We have considered appellants' argument that the "forms and messages" disclosed in Exhibit A correspond to "input and output messages or documents" as part of CBL [brief, page 8, emphasis added]. However, we note that appellants have

failed to provide a copy of the CBL first draft to be considered as evidence of actual reduction to practice.

We recognize that an accompanying exhibit need not support all claimed limitations, provided that any missing limitation is supported by the declaration itself. Ex parte Ovshinsky, 10 USPQ2d 1075 (Bd. Pat. App. & Inter. 1989) [emphasis added]. In the instant case, we note that appellants have also failed to explain how dependent claims 2-16 and 62-72 are supported by Exhibit A. In the alternative, appellants have failed to provide support for the missing dependent claim limitations in the declaration itself, so as to conclusively show possession of the complete instant claimed invention before the critical date. For at least the aforementioned reasons, we conclude that the character and weight of the evidence submitted pursuant to 37 C.F.R. § 1.131(b) is insufficient to show actual reduction to practice before the critical date of Sept. 1998. Accordingly, we consider the McKendrick reference as prior art, *infra.*

II. We consider next the examiner's rejection of claims 1-16 and 61-72 as being unpatentable over the teachings of McKendrick in view of W3C [answer, page 3].

In rejecting claims under 35 U.S.C. § 103, it is incumbent upon the examiner to establish a factual basis to support the legal conclusion of

obviousness. See In re Fine, 837 F.2d 1071, 1073, 5 USPQ2d 1596, 1598 (Fed. Cir. 1988). In so doing, the examiner is expected to make the factual determinations set forth in Graham v. John Deere Co., 383 U.S. 1, 17, 148 USPQ 459, 467 (1966). The examiner must articulate reasons for the examiner's decision. In re Lee, 277 F.3d 1338, 1342, 61 USPQ2d 1430, 1433 (Fed. Cir. 2002). In particular, the examiner must show that there is a teaching, motivation, or suggestion of a motivation to combine references relied on as evidence of obviousness. Id. 277 F.3d at 1343, 61 USPQ2d at 1433-34. The examiner cannot simply reach conclusions based on the examiner's own understanding or experience - or on his or her assessment of what would be basic knowledge or common sense. Rather, the examiner must point to some concrete evidence in the record in support of these findings. In re Zurko, 258 F.3d 1379, 1386, 59 USPQ2d 1693, 1697 (Fed. Cir. 2001). Thus the examiner must not only assure that the requisite findings are made, based on evidence of record, but must also explain the reasoning by which the findings are deemed to support the examiner's conclusion. However, a suggestion, teaching, or motivation to combine the relevant prior art teachings does not have to be found explicitly in the prior art, as the teaching, motivation, or suggestion may be implicit from the prior art as a whole, rather than expressly stated in the references. The test for an implicit showing is what the combined teachings, knowledge of one of

ordinary skill in the art, and the nature of the problem to be solved as a whole would have suggested to those of ordinary skill in the art. In re Kahn, 441 F.3d 977, 987-88, 78 USPQ2d 1329, 1336 (Fed. Cir. 2006) citing In re Kotzab, 217 F.3d 1365, 1370, 55 USPQ2d 1313, 1316-17 (Fed. Cir. 2000). See also In re Thrift, 298 F. 3d 1357, 1363, 63 USPQ2d 2002, 2008 (Fed. Cir. 2002). These showings by the examiner are an essential part of complying with the burden of presenting a prima facie case of obviousness. Note In re Oetiker, 977 F.2d 1443, 1445, 24 USPQ2d 1443, 1444 (Fed. Cir. 1992). If that burden is met, the burden then shifts to the applicant to overcome the prima facie case with argument and/or evidence. Obviousness is then determined on the basis of the evidence as a whole and the relative persuasiveness of the arguments. See Id.; In re Hedges, 783 F.2d 1038, 1039, 228 USPQ 685, 686 (Fed. Cir. 1986); In re Piasecki, 745 F.2d 1468, 1472, 223 USPQ 785, 788 (Fed. Cir. 1984); and In re Rinehart, 531 F.2d 1048, 1052, 189 USPQ 143, 147 (CCPA 1976).

As per claims 1-16

Since appellants' arguments with respect to this rejection have treated these claims as a single group which stand or fall together, we will consider independent claim 1 as the representative claim for this rejection. See 37 C.F.R. § 41.37(c)(1)(vii)(2004).

At the outset, we note that Appellants have made an admission in the reply brief at page 1, ¶4:

> This reply focuses on the McKendrick reference, because the second WSCI [*sic*, "W3C"] reference serves only to connect XML to the claim wording, "respective descriptions of sets of storage units and logical structures for the sets of storage units." Appellants and Respondent agree that this wording reads on XML and XML reads on this wording [emphasis added].

Thus, appellants have conceded that W3C teaches "respective descriptions of sets of storage units and logical structures for the sets of storage units," as claimed [reply brief, page 1, ¶4; claims 1 and 61]. With respect to the remaining limitations of representative claim 1, appellants argue that McKendrick's disclosure of the use of XML defines only one document, and does not disclose an interface to a transaction [brief, pages 9 and 10]. Appellants argue that McKendrick's disclosure of purchase orders and invoices does not teach the claimed input and output documents and associated definitions [brief, page 10; claim 1]. Appellants further argue that McKendrick does not teach or enable a machine-readable specification of an interface, as claimed [brief, page 11, claim 1].

In response, the examiner notes that McKendrick discloses XML financial transactions on the Internet such as purchase orders and invoices [answer, page 20; McKendrick, pages 1-2]. The examiner asserts that

McKendrick implies an interface for business transactions since the online business transactions must be performed with some type of interface to process the purchase orders and invoices [*id.*]. The examiner corresponds McKendrick's purchase orders and invoices to the instant claimed input and output documents [*id.*]. The examiner argues that the purchase orders and the invoices include definitions because they are XML documents. The examiner relies upon W3C as teaching the recited "definitions of the input and output documents comprising respective descriptions of sets of storage units and logical structures for the sets of storage units" [*id.*, claim 1]. The examiner concludes that the modification of McKendrick with W3C would provide the specific syntax needed to implement an XML structure as a mechanism to define the logical structures and support the use of storage units utilized in business documents to effectively perform business services [answer, page 20].

We note that McKendrick teaches a web site interface for performing "T. Rowe Price" financial transactions as shown in the illustration on page 2 of the reference. We find that the instant claimed "interface for transactions among nodes in a network" clearly reads upon this disclosure, noting that the Internet includes "a plurality of nodes" and also that web pages are stored on a computer readable medium, as claimed [claim 1]. We agree with the examiner that the instant claimed input and output documents

broadly read upon McKendrick's XML purchase orders and invoices [see last paragraph on page 2]. In particular, we note that a purchase order is clearly an input document from the perspective of a business providing products or services to customers over the Internet. We further note that an invoice is an output document when that same business bills the customer for the provided product or service. We find that the McKendrick's use of XML (as defined by the W3C XML specification) to perform financial transactions on the Internet clearly meets the language of the claim that recites "a machine readable specification of an interface to transaction processes stored in memory accessible by at least one node in the network, including interpretation information providing a definition of an input document, and a definition of an output document" [claim 1]. We note that XML is a machine readable specification that allows the author to define his own tags and his own document structure [see W3C, p. 13, ¶3 "Logical Structures"]. Because a "well-formed" XML document is a document that conforms to XML syntax rules, we find that McKendrick's XML purchase orders and invoices are clearly associated with corresponding XML declarations (i.e., definitions) [see e.g., W3C, p. 8, ¶2.8]. We note that appellants have admitted *supra* that the W3C reference teaches the remaining limitations of "respective descriptions of sets of storage units and logical structures for the sets of storage units," as claimed [reply brief, page 1, ¶4; claim 1]. Therefore, we

agree with the examiner that the combination of McKendrick and W3C teaches all that is claimed.

Hindsight

Appellants further argue that the examiner has impermissibly relied upon hindsight in formulating the rejection [brief, pages 10 and 11; reply brief, page 8].

We disagree. We note that the Court of Appeals for the Federal Circuit has determined that the motivation to combine under §103 must come from a teaching or suggestion within the prior art, within the nature of the problem to be solved, or within the general knowledge of a person of ordinary skill in the field of the invention, to look to particular sources, to select particular elements, and to combine them as combined by the inventor. Ruiz v. A.B. Chance Co., 234 F.3d 654, 665, 57 USPQ2d 1161, 1167 (Fed. Cir. 2000) [emphasis added]. In the instant case, we find that the use of the W3C XML specification (and associated XML definitions and syntax) is inherent in McKendrick's XML-based web pages and transaction documents. We further find that because XML is a recognized standard, such use of XML would have been well within the general knowledge of a person of ordinary skill in the field of the invention. Accordingly, we conclude that the examiner has sufficiently explained why an artisan

possessing knowledge of McKendrick and W3C at the time of the invention would have been motivated to look to W3C, to select particular elements, and to combine them with McKendrick. Accordingly, we find no evidence of record to support appellants' assertion that the examiner has impermissibly used hindsight in formulating the rejection.

Therefore, we agree with the examiner that McKendrick and W3C teach all the limitations disputed by appellants in the briefs. Accordingly, we will sustain the examiner's rejection of representative claim 1. We note that dependent claims 2-16 fall with independent claim 1 since appellants have not separately argued the patentability of these claims. See In re Nielson, 816 F.2d 1567, 1572, 2 USPQ2d 1525, 1528 (Fed. Cir. 1987). See also 37 C.F.R. § 41.37(c)(1) (vii)(2004). Accordingly, we will also sustain the examiner's rejection of these claims for the reasons set forth by the examiner in the rejection.

As per claims 61-72

Since appellants' arguments with respect to this rejection have treated these claims as a single group which stand or fall together, we will consider independent claim 61 as the representative claim for this rejection. See 37 C.F.R. § 41.37(c)(1)(vii)(2004).

Appellants argue: "the Examiner's words that McKendrick 'does not disclose' what is claimed make a good argument for reversal" [brief, page 12]. Appellants further argue that the mention of purchase orders and invoices as potential future uses for XML is an improper basis for a §103 rejection [brief, page 12, emphasis added].

We note that the examiner states in the rejection of claim 61 that McKendrick does not explicitly disclose that the definitions of the input document and the output document comprise respective descriptions of sets of storage units and logical structures for the sets of storage units [answer, page 4].

The Court of Appeals for the Federal Circuit has determined that one cannot show nonobviousness by attacking references individually where the rejections are based on combinations of references. In re Merck & Co., Inc., 800 F.2d 1091, 231 USPQ 375 (Fed. Cir. 1986). In the instant case, the examiner's rejection is based upon the combination of the McKendrick and W3C references. We have previously addressed the issue of input and output documents and associated XML definitions [see the discussion of claim 1 *supra*]. We note that McKendrick teaches the notoriously well known use of purchase orders and invoices [page 2, last paragraph] and both McKendrick and W3C disclose the use of XML. With respect to appellants' argument that McKendrick merely discloses potential future uses for XML,

we find that McKendrick provides a compelling suggestion to an artisan that it would be advantageous to implement purchase orders and invoices on the Internet using XML [see McKendrick, page 2, last paragraph, e.g., "XML will allow a rich array of business applications to be implemented"; see also brief, page 12]. We note again that appellants have conceded that the W3C reference teaches the remaining limitations of "respective descriptions of sets of storage units and logical structures for the sets of storage units," as claimed [reply brief, page 1, ¶4; claim 61]. Therefore, we agree with the examiner that the combination of McKendrick and W3C teaches all that is claimed.

Appellants restate their argument that the examiner has impermissibly relied upon hindsight in formulating the rejection [brief, page 12; reply brief, page 9]. We note that we have fully addressed this issue with respect to the rejection of claim 1, as discussed *supra*.

Therefore, for at least the aforementioned reasons, we find that McKendrick, as modified by W3C, teaches all the limitations disputed by appellants in the briefs with respect to representative claim 61. Accordingly, we will sustain the examiner's rejection of independent claim 61. We note that dependent claims 62-72 fall with claim 61 since appellants have not separately argued the patentability of these claims. Therefore, we will

sustain the examiner's rejection of these claims for the reasons set forth by the examiner in the rejection.

In summary, we have sustained the examiner's rejection of all claims on appeal. Therefore, the decision of the examiner rejecting claims 1-16 and 61-72 is affirmed.

No time period for taking any subsequent action in connection with this appeal may be extended under 37 C.F.R. § 1.136(a)(1)(iv).

AFFIRMED.

Errol A. Krass)
Administrative Patent Judge)
)
)
Jerry Smith) BOARD OF PATENT
Administrative Patent Judge) APPEALS AND
)
Howard B. Blankenship) INTERFERENCES
Administrative Patent Judge)
)

JS/sjc/kis

HAYNES, BEFFEL & WOLFELD, L.L.P.
P. O. BOX 366
HALF MOON BAY, CA 94019

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS
AND INTERFERENCES

Ex parte BART ALAN MELTZER , TERRY ALLEN,
MATTHEW DANIEL FUCHS, ROBERT JOHN GLUSHKO,
and MURRAY MALONEY

Appeal 2006-1639
Application 09/173,858
Technology Center 2100

Decided: May 21, 2007

Before HOWARD B. BLANKENSHIP, ALLEN R. MACDONALD,
and ST. JOHN COURTENAY III, *Administrative Patent Judges*.

COURTENAY, *Administrative Patent Judge*.

ON REQUEST FOR REHEARING

Appellants request that we reconsider our decision of Aug. 31, 2006, in which we sustained the rejection of claims 1-16 and 61-72 as unpatentable under 35 U.S.C. § 103.

We have reconsidered our Decision of Aug. 31, 2006, in light of Appellants' comment in the request for rehearing, and we find no errors therein. We decline to change our prior Decision for the following reasons:

Appellants argue the Board raises for the first time the issue of the first draft of CBL (Common Business Language) not being submitted with the Rule 131 Declaration filed on Jan. 31, 2005.[1] Appellants argue that if the Examiner had ever raised this as a factual issue, a supplemental declaration could have readily been submitted.

We note that it is Appellants' responsibility to timely submit all evidence necessary to support a Rule 131 declaration. Exhibit A clearly discloses the creation of a first draft of CBL, but Appellants chose not to provide the first draft of CBL with the Rule 131 declaration. The record shows the Examiner, as finder of fact, determined that Appellants' evidence was insufficient to support the Rule 131 declaration (*see* pp. 5-6 of the Office Action mailed on April 18, 2005). We find the prosecution history indicates that Appellants had ample opportunity to provide supplemental declarations as necessary, particularly in view of the fact that the Examiner reopened the prosecution by issuing a non-final office action on April 18, 2005. We found in our Decision that Appellants failed to point to specific portions of Exhibit A to demonstrate actual reduction to practice of specific claim terms (Decision 8, ¶ 2).

In reconsidering the single page of Exhibit A, we again find the listed technical activities 1-6 fail to provide a factual showing that the embodiment relied upon actually worked for its intended purpose (Exhibit A).

Specifically, we find that technical activity 1 merely provides evidence of development of a design *philosophy*. We find that technical activities 2-4 merely disclose *analysis* of: (a) *existing standards*, (b) *proposed* metadata frameworks for Internet resources, and (c) *semantics* of commerce, and *analysis* of certain *proposals* (e.g., "Open Buying on the Internet specification"). We have fully addressed the issue of technical activity 5 *supra* (i.e., creation of a first draft of Common Business Language (CBL)). Lastly, we find that technical activity 6 merely discloses *determining an approach* for CBL support. With respect to the second set of activities (1-5) shown at the bottom of Exhibit A, we note that these activities merely refer to activities *planned* for the second quarter of 1998. We find no evidence of record that these planned activities actually occurred. Thus, we find Appellants have failed to show prior "possession of either the whole invention claimed or something falling within the claim, in the sense that the claim as a whole reads on it." *See In re Tanczyn*, 347 F.2d 830, 833, 146 USPQ 298, 301 (CCPA 1965). Likewise, we find that Appellants have failed to show "priority with respect to so much of the claimed invention as the reference happens to show." *See In re Stempel,* 241 F.2d 755, 759-60, 113 USPQ 77, 81 (CCPA 1957).

With respect to the issue of antedating the subject matter of the dependent claims, we note that the plain language of 37 C.F.R. § 1.131(a) is directed to the submission of an appropriate oath or declaration to establish

[1] *See* 37 C.F.R. §1.131.

invention of the subject matter *of the rejected claim*, and not to the claims as a whole (emphasis added), as shown below.

> (a) When *any claim* of an application or a patent under reexamination is rejected, the inventor of the subject matter of the rejected claim, the owner of the patent under reexamination, or the party qualified under §§ 1.42, 1.43, or 1.47, may submit an appropriate oath or declaration to establish invention of the subject matter *of the rejected claim* prior to the effective date of the reference or activity on which the rejection is based [emphasis added].

(37 C.F.R. § 1.131(a), revised, 69 FR 56481, Sept. 21, 2004, effective Oct. 21, 2004).

Even assuming *arguendo* that our reading of the rule is too literal (i.e., given that Appellants are not required to provide an exhibit that supports all the claim limitations),[2] we nevertheless find the text of Appellants'

[2] In order to remove a reference, a Rule 131 declaration must show that prior to the effective date of the reference the applicant had reduced to practice so much of the claimed invention as the reference shows. *In re Scheiber,* 587 F.2d 59, 61-62, 199 USPQ 782, 784 (CCPA 1978); *In re Stempel,* 241 F.2d at 759-60, 113 USPQ at 81 (CCPA 1957) ("We are convinced that under the law all the applicant can be required to show is priority with respect to so much of the claimed invention as the reference happens to show. When he has done that he has disposed of the reference."). *cf. In re Tanczyn*, 347 F.2d at 833, 146 USPQ at 301 (CCPA 1965) ("We never intended by the language used in Stempel to authorize the overcoming of references by affidavits showing that the applicant had invented, prior to the reference date, a part, some parts, or even a combination of parts, used to create an embodiment of his claimed invention, where the part or parts are

Declaration fails to make up for the deficiencies of Exhibit A. In particular, we find Appellants have failed to clearly explain how the specific elements of Exhibit A (e.g., the subsumed registry and associated query services) correspond to (or render obvious) specific elements of the instant claimed invention, as enumerated in our Decision on page 8. Likewise, we find Appellants have failed to show a reduction to practice of at least as much of the claimed invention as the McKendrick reference shows. *See In re Stempel,* 241 F.2d at 759-60, 113 USPQ at 81.

In particular, we note that a registry, per se, is not recited in any of instant claims 1-16 or 61-72. We fail to see how a registry (i.e., a place where a register is kept) is equivalent to the recited "interface for transactions" (claim 1). We disagree with Appellants' sweeping assertion that one of ordinary skill having knowledge of the Declaration (at ¶ 4) and Exhibit A would have understood the registry disclosed in Exhibit A to include interface definition data structures having input and output document schemas (*see* Request for Rehearing, p. 10, § 3). We further note that Appellants correspond the "repository in memory" recited in dependent claim 4 with the "registry" of Exhibit A (*see* Request for Reconsideration, p. 13, note 3). In contrast, we note that a repository in memory more closely corresponds with the "storage units" that are associated with the "interface

---

not within the scope of the claims being sought ... It is not sufficient to show in a Rule 131 affidavit that an invention wholly outside of that being claimed was made prior to the reference date.").

for transactions" (claim 1). As pointed out in our Decision at page 12, Appellants made an admission in the Reply Brief:

> This reply focuses on the McKendrick reference, because the second WSCI [*sic*, "W3C"] reference serves only to connect XML to the claim wording, "respective descriptions of sets of storage units and logical structures for the sets of storage units." *Appellants and Respondent agree that this wording reads on XML and XML reads on this wording* [emphasis added]. (Reply Br. 1, ¶ 4).

Thus, we find Appellants have acknowledged that the secondary W3C XML reference teaches "respective descriptions of sets of storage units and logical structures for the sets of storage units," as claimed (Reply Br. 1, ¶4; claims 1 and 61).

Appellants argue that simply using XML documents does not teach an interface definition data structure that pairs input and output XML documents as a process interface (Request for Rehearing 14).

We note again that McKendrick teaches a web site *interface* for performing "T. Rowe Price" financial transactions as shown in the illustration on page 2 of the reference. Therefore, we find the instant claimed "interface for transactions among nodes in a network" is taught and/or suggested by McKendrick. We note that the Internet includes "a plurality of nodes" and also that web pages are stored on a computer readable medium, as claimed (claim 1). Therefore, we agree with the Examiner that the instant claimed *input* and *output* documents are taught by McKendrick's use of XML *purchase orders* and *invoices* (*see* McKendrick, last paragraph on page 2). We note again that a *purchase order* is clearly an *input document* from the perspective of a business providing products or

services to customers over the Internet. We further note that an *invoice* is an *output document* when that same business bills the customer for the provided product or service. Therefore, we find that McKendrick's use of XML (as defined by the W3C XML specification) to perform financial transactions on the Internet clearly meets the language of the claim that recites "a machine readable specification of an interface to transaction processes stored in memory accessible by at least one node in the network, including interpretation information providing a definition of an input document, and a definition of an output document" (claim 1). With respect to Appellants' argument that neither McKendrick nor W3C teaches a "Web services interface," we note that a "Web services Interface" is not claimed (Request for Rehearing 14). Accordingly, for at least the aforementioned reasons, we agree with the Examiner that the combination of McKendrick and W3C teaches and/or suggests all that is claimed.

Pursuant to 37 C.F.R. ¶ 41.52(a)(1), we have no discretion to consider the new evidence and associated new arguments submitted by Appellants with the Request for Rehearing.

> Arguments not raised in the briefs before the Board and evidence not previously relied upon in the brief and any reply brief(s) are not permitted in the request for rehearing except as permitted by paragraphs (a)(2) and (a)(3) of this section. (37 C.F.R. ¶ 41.52(a)(1)).

We note that if Appellants wish to have the newly presented evidence considered by the Examiner, the proper procedure is to file a Request for Continued Examination (RCE) under 37 C.F.R. § 1.114.

We have carefully considered the arguments raised by Appellants in the Request for Rehearing, but none of these arguments are persuasive that our original Decision was in error. We are still of the view, that the invention set forth in claims 1-16 and 61-72 is unpatentable over the applied prior art based on the record before us in the original appeal. This Decision on Appellants' Request for Rehearing is deemed to incorporate our earlier Decision (mailed Aug. 31, 2006) by reference. *See* 37 C.F.R. § 41.52(a)(1).

We have granted Appellants' request to the extent that we have reconsidered our Decision of Aug. 31, 2006, but we deny the request with respect to making any changes therein.

REHEARING DENIED

KIS

HAYNES, BEFFEL & WOLFELD LLP
P. O. BOX 366
HALF MOON BAY, CA 94019

OIN 1004-1
EJB

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Ex parte Meltzer et al.

Application No. 09/173,858
Appeal No. 2006-1639

DECISION ON PETITION

This is a decision on a Petition Under Rule 183 For (1) Second Rehearing With Suggestion For Expanded Panel; and (2) Oral Argument, filed July 23, 2007. The Petition requests a second rehearing with an expanded panel and oral argument and, on the merits, reversal of the claim rejections. Alternatively, the Petition requests that the application be remanded to the examiner for consideration of newly presented evidence. The Petition will be treated as a petition to the Chief Administrative Patent Judge under 37 C.F.R. § 41.3.

FINDINGS OF FACT

1. On August 31, 2006, the Board of Patent Appeals and Interferences (Board) entered a Decision (Original Decision) affirming the rejection of the claims on appeal.

2. On October 30, 2006, Appellants filed a Request for Rehearing, including newly presented evidence and associated new arguments.

3. On May 21, 2007, the Board entered a Decision on Rehearing denying Appellants' request to change the Original Decision and stating that "[p]ursuant to 37 C.F.R. ¶ 41.52(a)(1), we have no discretion to consider the new evidence and associated new arguments submitted by Appellants with the Request for Rehearing." Decision on Rehearing at 7. The Board also noted that "if Appellants wish to have the newly presented evidence considered by the Examiner, the proper procedure is to file a Request for Continued Examination (RCE) under 37 C.F.R. § 1.114." *Id.*

4. On July 23, 2007, at the same time as filing this Petition, Appellants also filed a Request for Continued Examination Following Appeal along with the evidence newly presented with the Request for Rehearing and additional newly presented evidence.

DISCUSSION

MPEP § 706.07(h)(XI)(A) states:

> The filing of an RCE (accompanied by the fee and a submission) after a decision by the Board of Patent Appeals and Interferences, but before the filing of a Notice of Appeal to the Court of Appeals for the Federal Circuit (Federal Circuit) or the commencement of a civil action in federal district court, will also result in the finality of the rejection or action being withdrawn and the submission being considered.

Even if an RCE is considered conditional upon some other action, "the Office will treat the 'conditional' RCE and payment as if an RCE and payment of the fee set forth in 37 CFR 1.17(e) had been filed." MPEP § 706.07(h)(III)(C).

Thus, to the extent that Appellants may consider the RCE conditional upon denial of the Petition, it is clear that the RCE takes precedence over the Petition, rendering the latter moot.

DECISION

For the foregoing reasons, the Petition is **DISMISSED** as moot and the application is returned to the Examiner for processing of the RCE.

Michael R. Fleming
Chief Administrative Patent Judge
Board of Patent Appeals and Interferences

Ernest J. Beffel, Jr.
Haynes Beffel & Wolfeld LLP
P.O. Box 366
Half Moon Bay, CA 94019

Telephone: (650) 712-0340
Facsimile: (650) 712-0263

APPENDIX XII.
EVIDENCE

I. **Office Action**

Office Action Setting Out Rejection on Appeal dated October 9, 2008

Original Application Referenced in OA dated October 16, 1998

OIN 1004-1
EJB

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/173,858	10/16/1998	BART ALAN MELTZER	OIN 1004-1	4734

22470 7590 10/09/2008
HAYNES BEFFEL & WOLFELD LLP
P O BOX 366
HALF MOON BAY, CA 94019

EXAMINER	
HUYNH, CONG LAC T	
ART UNIT	PAPER NUMBER
2178	
MAIL DATE	DELIVERY MODE
10/09/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

HAYNES BEFFEL & WOLFELD LLP

DOCKETED 10-15-08 BY HK
ACTION US-OFFICE ACTION (3 MONTH)
DUE DATES 1-9-09 DUE
4-9-09 FINAL
DOCKET NO. OIN 1004-1

HAYNES BEFFEL & WOLFELD LLP
OCT 14 2008
RECEIVED

R-36

Office Action Summary	Application No.	Applicant(s)	
	09/173,858	MELTZER ET AL.	
	Examiner	Art Unit	
	Cong-Lac Huynh	2178	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *22 July 2008*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-16 and 61-74* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-16, 61-74* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☒ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date *7/23/08*.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This action is responsive to: RCE filed on 7/21/08 to the application filed on 10/16/98.

2. Claims 73-74 are added.

3. Claims 1-16, 61-74 are pending in the case. Claims 1, 61, 73 are independent claims.

Claim Rejections - 35 USC § 112

4. The following is a quotation of the first paragraph of 35 U.S.C. 112:

> The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same and shall set forth the best mode contemplated by the inventor of carrying out his invention.

5. Claims 73-74 are rejected under 35 U.S.C. 112, first paragraph, as failing to comply with the enablement requirement. The claim(s) contains subject matter which was not described in the specification in such a way as to enable one skilled in the art to which it pertains, or with which it is most nearly connected, to make and/or use the invention.

Independent claim 73 is directed to specification of an interface to an operation rather than an interface to plural transaction processes. Applicants point out the support in the specification at pages 17, 25-27, and 86.

However, as disclosed in these pages, particularly page 25, lines 25-26, page 26, lines 7-9, 43-48, page 27, lines 1-2, 9-12, when the service for only an input document or only an output document, then an operation is performed. But when the service is for both an input document and an output document as in the claim, that means "by way of

input and output documents", then multiple operations should be performed for input and output documents. The claimed limitation, therefore, is not consistent with the specification.

Dependent claim 74 is rejected for fully incorporating the deficiencies of their base claim.

6. The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

7. Claims 73-74 are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

Regarding independent claim 73, it is unclear how an operation is performed for two processes for an input document and an output document.

Dependent claim 74 is rejected for fully incorporating the deficiencies of their base claim.

Claim Rejections - 35 USC § 103

8. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

9. This application currently names joint inventors. In considering patentability of the claims under 35 U.S.C. 103(a), the examiner presumes that the subject matter of the various claims was commonly owned at the time any inventions covered therein were made absent any evidence to the contrary. Applicant is advised of the obligation under 37 CFR 1.56 to point out the inventor and invention dates of each claim that was not commonly owned at the time a later invention was made in order for the examiner to consider the applicability of 35 U.S.C. 103(c) and potential 35 U.S.C. 102(e), (f) or (g) prior art under 35 U.S.C. 103(a).

10. Claims 1-16, 61-72 remain rejected under 35 U.S.C. 103(a) as being unpatentable over McKendrick, *Banks begin to play with XML*, Bank Technology News, Sep 1998, Vol. 11, Iss. 9, pg. 6, 2 pgs, in view of W3C, *Extensible Markup Language (XML) 1.0*, 2/10/98, pages 1-37 (from the IDSs).

Regarding independent claim 1, McKendrick discloses:

- a machine-readable specification of an interface to transaction processes stored in memory accessible by at least one node in the network, including interpretation information providing a definition of an input document, and a definition of an output document (pages 1-2: McKendrick discloses applying XML in financial area to provide better bank services and utilizing XML for on-line business transactions involved with manipulation and transfer of data in the Internet such as purchase orders, invoices, and customer information. The purchase orders are considered as input documents, and the invoices are

considered as output documents of the purchase orders in business transactions. Since the purchase orders as well as the invoices, which are the input and output documents, are in XML, they definitely include information providing the definition for such a document according to XML structures. And since the transaction documents are in XML format, these documents are machine-readable documents and should be stored in memory of a server accessible by at least one node in the network)

McKendrick does not explicitly disclose that the definitions of the input document and the output document comprising respective descriptions of sets of storage units and logical structures for the sets of storage units.

W3C discloses that each XML document comprises respective descriptions of set of storage units and logical structures for the set of storage units (page 3, Introduction: "XML documents are made up of *storage units* called entities, which contain either *parsed or unparsed data*. Parsed data is made up of characters, some of which form character data, and some of which form *markup*. *Markup encodes a description of the document's storage layout and logical structure*. XML provides a mechanism to impose constraints on the storage layout and logical structure.")

It would have been obvious to one of ordinary skill in the art at the time of the invention was made to have combined McKendrick into W3C for the following reason. McKendrick discloses the transaction documents such as the purchase orders and the invoices in *XML format* for a business transaction over the Internet where a user can search and buy an item on-line, and W3C discloses the *structures of an XML document*

which comprises storage units and the logical structures for the set of storage units. This motivates to combine W3C into McKendrick for supporting the business transaction documents in XML format using the XML characteristics disclosed in W3C.

Regarding claim 2, which is dependent on claim 1, McKindrick does not disclose that the interpretation information includes data type specification for at least one logical structure in the definitions of the input and output document.
W3C discloses that each XML document contains one or more elements which are delimited by starts-tags and end-tags, and each element has a *type* identified by name called generic identifier and may have a set of *attribute specification* (page 13, Logical structure).
It would have been obvious to one of ordinary skill in the art at the time of the invention was made to have combined McKendrick into W3C for supporting the business transaction documents in XML format using the XML characteristics disclosed in W3C.

As mentioned in claim 1, since the documents used in the purchase transaction in McKindrick are in XML format, these documents inherit the features of a general XML document as disclosed in W3C. This is applied for all the claims relating to the transaction document structures and W3C is used for rejecting.

Regarding claim 3, which is dependent on claim 1, W3C discloses that the interpretation information includes at least one data structure mapping predefined sets of storage

units for a particular logical structure in the definition of the input and output documents, to respective entries in a list (pages 14-17).

Regarding claims 4 and 5, which are dependent on claim 1, McKindrick and W3C do not disclose explicitly that a repository in memory accessible by at least one node in the network storing a library of logical structures, interpretation information for logical structures, and the identifier of a transaction. However, it would have been obvious to one of ordinary skill in the art at the time of the invention was made to have modified McKindrick and W3C to include a repository in memory for storing logical structures and the identifier of a transaction interface since it was well known in the art that any defined data for a program in a network should have a name for identifying and should be stored in a memory of a server for using later on such as retrieving data, identifying data, or manipulating data.

Regarding claim 6, which is dependent on claim 1, W3C discloses that the machine readable specification includes a document compliant with a definition of an interface document including logical structures for storing an identifier of the interface, and for storing at least one of specifications and references to specifications of a set of one or more transactions supported by the interface (page 13).

Regarding claim 7, which is dependent on claim 6, McKindrick does not disclose a reference to a specification of a particular transaction, and the specification of the

particular transaction includes a document including logical structures for storing at least one of definitions and references to definitions input and output documents for the particular transaction. Instead, McKindrick discloses *applying XML for business-to-business transaction where data such as purchase orders and invoices are manipulated and transferred over the Internet* (page 2).

W3C discloses that each XML document comprises respective descriptions of set of storage units and logical structures for the set of storage units (page 3, Introduction: "XML documents are made up of *storage units* called entities, which contain either *parsed or unparsed data*. Parsed data is made up of characters, some of which form character data, and some of which form *markup*. *Markup encodes a description of the document's storage layout and logical structure*. XML provides a mechanism to impose constraints on the storage layout and logical structure.")

It would have been obvious to one of ordinary skill in the art at the time of the invention was made to have combined W3C into McKindrick to include a reference to a specification of a particular transaction which has logical structures for storing at least one of definitions and references to documents as in W3C for the particular business transaction as in McKindrick since a reference is considered as a name or an identifier and the transaction documents in McKindrick such as the purchase orders and the invoices, considered as the input and output documents, must have a document name for identifying purpose.

Regarding claim 8, which is dependent on claim 1, W3C discloses that the storage units comprise parsed data (page 3, Introduction: "XML documents are made up storage units called entities, which contain either parsed or unparsed data...").

Regarding claim 9, which is dependent on claim 1, McKindrick does not explicitly disclose the parsed data in at least one of the input and output documents comprises:

- character data encoding text characters in the one of the input and output document
- markup data identifying sets of storage units according to the logical structure of the one of the input and output documents

Instead McKindrick discloses the business transactions involved with manipulation and transfer data such as purchase orders and invoices where invoices are considered as the output documents produced from the data portion of the purchase orders, which are considered as the input document (pages 1-2).

W3C discloses that the parsed data comprises:

- character data encoding text characters in XML documents (page 3, Introduction: "*XML documents* are made up storage units ...*Parsed data* is made up characters, some of which form *character data* ..."; page 6, Characters: "A parsed entity contains text, a sequence of characters, which may represent markup or character data
- markup data identifying sets of storage units according to the logical structure of XML documents (page 3, Introduction: "*XML documents* are made up storage

units ... *Parsed data* is made up characters, some of which form character data, and some of which form *markup*. *Markup encodes a description of the document's storage layout and logical structure* ...")

It would have been obvious to one of ordinary skill in the art at the time of the invention was made to have combined W3C into McKindrick since the XML business documents in McKindrick which function as input and output documents should comprise parsed data with claimed features since these features are characteristics of an XML document as taught in W3C.

Regarding claim 10, which is dependent on claim 9, W3C discloses that at least one of the sets of storage units encodes a plurality of text characters providing a natural language word (page 6, Document, page 7, Characters and page 8, Character Data and Markup: since the storage units encodes by character data and markup which are *text*, the storage units provide a natural language word).

Regarding claim 11, which is dependent on claim 8, W3C discloses that the interpretation information for at least one of the sets of storage units identified by a particular logical structure of at least one of the input and output documents, encodes respective definitions for sets of parsed characters (page 9: "the function of the markup in an XML document is to describe its storage and logical structure and to associate attribute-value pairs with its logical structures. XML provides a mechanism, the document type declaration, to *define constraints on the logical structure* and to support

the use of predefined storage units ... the XML document type declaration contains or points to markup declarations that provide a grammar for a class of documents. This grammar is known as a *document type definition, or DTD ...*").

Regarding claim 12, which is dependent on claim 8, W3C discloses that the storage units comprise unparsed data (page 3, Introduction: "XML documents are made up storage units called entities, which contain either parsed or unparsed data..." page 20, Physical Structures).

Regarding claim 13, which is dependent on claim 1, as mentioned in claims 4 and 5 above, it would have been obvious to one of ordinary skill in the art at the time of the invention was made to have modified McKindrick and W3C to include a repository in memory for storing all data related to the purchase transactions since it was well known in the art that any defined data for a program in a network should be stored in a memory of a server for using later on such as retrieving data, identifying data, or manipulating data.

Regarding claim 14, which is dependent on claim 13, W3C discloses that the repository of document types includes a document type for identifying participant process in the network (page 9: "XML provides a mechanism, the *document type declaration*, to define constraints on the logical structure and to support the use of predefined storage units").

Regarding claim 15, which is dependent on claim 1, W3C discloses that the definitions of the input and output documents comprise document type definitions compliant with a standard Extensible Markup Language XML (page 9: "XML provides a mechanism, the document type declaration, to define constraints on the logical structure and to support the use of predefined storage units ... the XML document type declaration contains or points to markup declarations that provide a grammar for a class of documents. This grammar is known as a document type definition, DTD ... the DTD fro a document consists of both subsets taken together").

Regarding claim 16, which is dependent on claim 1, W3C discloses that the machine readable data structure including interpretation information comprises a document organized according to a document type definition compliant with a standard Extensible Markup Language XML (page 9: an XML document is a machine readable data structure organized according to a DTD compliant with the standard Extensible Markup Language).

Regarding independent claim 61, McKindrick does not disclose explicitly:

- defining a machine readable definition of an input document for a node in the network including resources to execute a process in the transaction, and a machine readable definition of an output document for the node, the definitions the input and output documents comprising respective descriptions of sets of storage units and logical structures for the sets of storage units

- providing interpretation information for the logical structures to the node

Instead McKindrick discloses applying XML in financial area to provide better bank services and utilizing XML for on-line business transactions involved with manipulation and transfer of data in the Internet such as purchase orders, invoices, and customer information (pages 1-2). The purchase orders in McKindrick are considered as input documents, and the invoices are considered as output documents of the purchase orders in business transactions. Since the purchase orders as well as the invoices, which are the input and output documents, are in XML format, they definitely include information to provide the definition for said documents according to XML structures. And since the transaction documents are in XML format, these documents are machine-readable documents and should be stored in memory of a server accessible by at least one node in the network.

W3C discloses:

- defining a machine readable definitions of documents comprising respective descriptions of sets of storage units and logical structures for the sets of storage units (page 3, Introduction and page 9: XML documents are made up of storage units which contain either parsed or unparsed data where parsed data is made up characters some of which form character data, and some of which form markup to encode a *description of the document storage layout and logical structures*).

- providing interpretation information for the logical structures (page 9: the function of the markup in an XML document is to associate *attribute-value* pairs with its logical structures)

It would have been obvious to one of ordinary skill in the art at the time of the invention was made to have combined McKendrick into W3C for the following reason. McKendrick discloses the transaction documents such as the purchase orders and the invoices in *XML format* for a business transaction over the Internet where a user can search and buy an item on-line and W3C discloses the *structures of an XML document* which comprises storage units and the logical structures for the set of storage units. This motivates to combine W3C into McKendrick for supporting the business transaction documents in XML format using the XML characteristics disclosed in W3C.

Claims 62-71 are for a method of claims 2-5, 8-12, 15, and are rejected under the same rationale.

Regarding claim 72, which is dependent on claim 61, McKindrick and W3C do not disclose:

- providing a parser to generate event signals in response to logical structures in the definition of the input document
- providing event listener program which respond to the event signals to execute the process

Instead McKindrick discloses the Internet business transactions via purchase orders and invoices in XML format where the purchase orders and the invoices are considered as input documents and output documents (pages 1-2).

It would have been obvious to one of ordinary skill in the art at the time of the invention was made to have modified McKindrick to include "providing a parser to generate event signals in response to logical structures..." and "providing event listener program which respond to the event signals to execute the process" for the following reason. The fact that McKindrick executes the transaction program by running the XML transaction documents which include logical structures suggests said parser and said event listener program as claimed, which are the must programs in the executing process.

11. Claims 73-74 are rejected under 35 U.S.C. 103(a) as being unpatentable over McKendrick, *Banks begin to play with XML*, Bank Technology News, Sep 1998, Vol. 11, Iss. 9, pg. 6, 2 pgs, in view of W3C, *Extensible Markup Language (XML) 1.0*, 2/10/98, pages 1-37 (from the IDSs).

Regarding independent claim 73, McKendrick discloses:

- a machine-readable specification of an interface to transaction processes stored in memory accessible by at least one node in the network, including interpretation information providing a definition of an input document, and a definition of an output document (pages 1-2: McKendrick discloses applying XML in financial area to provide better bank services and utilizing XML for on-line

business transactions involved with manipulation and transfer of data in the Internet such as purchase orders, invoices, and customer information. The purchase orders are considered as input documents, and the invoices are considered as output documents of the purchase orders in business transactions. Since the purchase orders as well as the invoices, which are the input and output documents, are in XML, they definitely include information providing the definition for such a document according to XML structures. And since the transaction documents are in XML format, these documents are machine-readable documents and should be stored in memory of a server accessible by at least one node in the network)

McKendrick does not explicitly disclose that the definitions of the input document and the output document comprising respective descriptions of sets of storage units and logical structures for the sets of storage units and the specification of an interface to an operation instead of plural transaction processes.

W3C discloses that each XML document comprises respective descriptions of set of storage units and logical structures for the set of storage units (page 3, Introduction: "XML documents are made up of *storage units* called entities, which contain either *parsed or unparsed data*. Parsed data is made up of characters, some of which form character data, and some of which form *markup*. *Markup encodes a description of the document's storage layout and logical structure*. XML provides a mechanism to impose constraints on the storage layout and logical structure.")

It would have been obvious to one of ordinary skill in the art at the time of the invention was made to have combined McKendrick into W3C for the following reason. McKendrick discloses the transaction documents such as the purchase orders and the invoices in *XML format* for a business transaction over the Internet where a user can search and buy an item on-line, and W3C discloses the *structures of an XML document* which comprises storage units and the logical structures for the set of storage units. This motivates to combine W3C into McKendrick for supporting the business transaction documents in XML format using the XML characteristics disclosed in W3C. Further, since McKendrick and W3C provide plural transaction processes, it is clear that one operation of one transaction is included in these transaction processes either for processing an input document or processing an output document.

Regarding claim 74, which is dependent on claim 73, McKindrick does not disclose that the interpretation information includes data type specification for at least one logical structure in the definitions of the input and output document.
W3C discloses that each XML document contains one or more elements which are delimited by starts-tags and end-tags, and each element has a *type* identified by name called generic identifier and may have a set of *attribute specification* (page 13, Logical structure).

It would have been obvious to one of ordinary skill in the art at the time of the invention was made to have combined McKendrick into W3C for supporting the business transaction documents in XML format using the XML characteristics disclosed in W3C.

Response to Arguments

12. Applicant's arguments filed 7/21/08 have been fully considered but they are not persuasive.

Applicants argue that "evidence other than declarations (the "previous evidence") was submitted but given no weight, due to the Examiner's mistaken invocation of res judicata" which can not be applied post-appeal to exclude evidence from consideration.

The Examiner respectfully disagrees.

The Examiner did consider the evidence but the evidence is not persuasive and not sufficient to remove the reference McKendrick. For example, the slide 30 as well all of the slides of Glushko's presentation are merely a PowerPoint document and do not show reduce to practice:

The slides presented on that date are merely a PowerPoint document for presentation. They are not an evidence of a complete product that is guaranteed that it worked with testing (see previous action, Response to Argurments)

Regarding Res Judicata, it is noted that after the Board's decision affirming the case on 8/31/06, no amendment was filed with the RCE. The claims in the RCE having no change are a duplicate of the claims presented earlier to the Board of Appeals.

In the previous office action, the submitted evidence with the RCE was considered but not sufficient, and so the reference can not be removed. The art rejection, therefore, is still the same as the rejection on the claims presented earlier to the Board of Appeal. Applying Res Judicata in the previous action, therefore, is proper. See MPEP 706.03 (w).

Applicant's statement that claim mapping is not required in MPEP. However, MPEP does require Applicants to show the possession of the invention:

> *The 37 CFR 1.131 affidavit or declaration must establish possession of either the whole invention claimed or something falling within the claim (such as a species of a claimed genus), in the sense that the claim as a whole reads on it. In re Tanczyn, 347 F.2d 830, 146 USPQ 298 (CCPA 1965)* (MPEP 715.02)

Claim mapping is one way to show such a possession. Claim mapping is for showing the match between what are claimed in the application and what Applicants truly invent.

Regarding the submitted declaration under rule 131, the inventor's statement in #20 that all versions of CBL before CBL 1.1 worked.

However, Exhibits D, E, G, H submitted as evidence of CBL are merely the coding files dated in 1997 and 1998 which are test files listed in Exhibit C without any statement recorded proving that these coding files run successfully :

> *"For an actual reduction to practice, the invention must have been sufficiently tested to demonstrate that it will work for its intended purpose, but it need not be in a commercially satisfactory stage of development. If a device is so simple,and its purpose and efficacy so*

> *obvious, construction alone is sufficient to demonstrate workability. King Instrument Corp. v. Otari Corp., 767 F.2d 853, 860, 226 USPQ 402, 407 (Fed. Cir. 1985). For additional cases pertaining to the requirements necessary to establish actual reduction to practice see DSL Dynamic Sciences, Ltd. v. Union Switch & Signal, Inc., 928 F.2d 1122, 1126, 18 USPQ2d 1152, 1155 (Fed. Cir. 1991) ("events occurring after an alleged actual reduction to practice can call into question whether reduction to practice has in fact occurred"); Corona v. Dovan, 273 U.S. 692, 1928 C.D. 252 (1928) ("**A process is reduced to practice when it is successfully performed.** A machine is reduced to practice when it is assembled, adjusted and used. A manufacture [i.e., article of manufacture] is reduced to practice when it is completely manufactured. A composition of matter is reduced to practice when it is completely composed." 1928 C.D. at 262-263 (emphasis added).); Fitzgerald v. Arbib, 268 F.2d 763, 765-66, 122 USPQ 530, 531-32 (CCPA 1959) ("the reduction to practice of a three-dimensional design invention requires the production of an article embodying that design" in "other than a mere drawing")" (MPEP 2138.05) (emphasis added)*

Also, as shown in Exhibit C, the article "Common Business Language (CBL)" dated December 5, 1997, the early version of CBL is unstable and merely a sketch, "*As of the date of this document's writing, this specification was still unstable, and the details of the linking attributes in the CBL DTDs should be considered as a sketch*" (emphasis added).

As mentioned in the Response to Arguments in the previous action, Dr. Glushko admitted in his article in December 1999 "How XML Enables Internet Trading Communities and Marketplaces" at http://www.infoloom.com/gcaconfs/WEB/philadelphia99/glushko.HTM that the early versions of CBL struggled with technical problems:

> *Work on CBL began in 1997, partly funded by a Department of Commerce's Advanced Technology Program research award on "Component-Based Commerce" to Veo Systems and three other firms [ATP]. Because of this* ***research pedigree, early versions of CBL strove for logical completeness, expressiveness, and compactness to test the abstract modeling power of XML for electronic commerce and to identify requirements for development tools and runtime support*** *(emphasis added)*

The Examiner's rejection regarding the early versions of CBL is not a conjecture, but based on one inventor's article. The statement in #20 of the Declaration, therefore, is not persuasive based on the fact of the inventor's article and the submitted Exhibits. Accordingly, it is clear that CBL version 1.1 is the first working version of CBL released in September 1998.

The submitted evidence does not prove actual reduction to practice prior September 1998, and so, there was no reduction to practice prior the effective date of the McKendrick reference.

Only the filling of a US patent application which complies with the disclosure requirement of 35 USC 112 constitutes a constructive reduction to practice. Accordingly, Applicants have not established prior invention. The rejection is maintained.

It is noted that evidence submitted in Exhibits A-J shows that CBL was developed since 1997 to overcome the impractical CORBA technology. That means the concept of CBL does exist since 1997. Proving concept coupled with due diligence may work more properly than proving reduce to practice before January 21, 1998 since early version of CBL is merely a sketch and admitted having technical troubles as mentioned above. Therefore, it is suggested to file a declaration to prove concept since 1997 coupled with due diligence since 1997 to the filing date of the application to swear behind an early date in 1997.

Conclusion

13. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

The prior art of record is listed on PTO 892.

14. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Cong-Lac Huynh whose telephone number is 571-272-4125. The examiner can normally be reached on Mon-Thurs (8:30-7:00).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Stephen Hong can be reached on 571-272-4124. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/Cong-Lac Huynh/
Primary Examiner, Art Unit 2178
09/30/08

Please type a plus sign (+) insi[illegible] box → ⊞

Approv[illegible]use through 09/30/00. OMB 0651-0032
Patent and Trademark O[illegible] U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number

UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Attorney Docket No.	19957.701	*Total Pages*	134

First Named Inventor or Application Identifier

Bart Alan Meltzer, Terry Allen, Matthew Daniel Fuchs, Robert John Glushko, Murray Maloney; Title: ***Documents for Commerce In Trading Partner Networks and Interface Definitions Based on the Documents***

Express Mail Label No.	EM083241789US

APPLICATION ELEMENTS
See MPEP chapter 600 concerning utility patent application contents.

ADDRESS TO: **Assistant Commissioner for Patents**
Box Patent Application
Washington, DC 20231

1. [ ] Fee Transmittal Form
(Submit an original, and a duplicate for fee processing)
2. [X] Specification [*Total Pages* 113]
(preferred arrangement set forth below)
- Descriptive title of the Invention
- Cross References to Related Applications
- Statement Regarding Fed sponsored R&D
- Reference to Microfiche Appendix
- Background of the Invention
- Brief Summary of the Invention
- Brief Description of the Drawings
- Detailed Description
- Claim(s)
- Abstract of the Disclosure

3. [X] Drawing(s) (37 CFR 1.152) [*Total Sheets* 16]

4. [X] Oath or Declaration (Unsigned) [*Total Pages* 4]

a. [ ] Newly executed (original or copy)

b. [ ] Copy from a prior application (37 CFR 1.63(d))
(for continuation/divisional with Box 17 completed)
[Note Box 5 below]
i. [ ] DELETION OF INVENTOR(S)
Signed statement attached deleting inventor(s) named in the prior application, see 37 CFR 1.63(d)(2) and 1.33(b).

5. [ ] Incorporation By Reference (*useable if Box 4b is checked*)
The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 4b, is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein.

6. [ ] Microfiche Computer Program (*Appendix*)

7. Nucleotide and/or Amino Acid Sequence Submission
(*if applicable, all necessary*)
a. [ ] Computer Readable Copy
b. [ ] Paper Copy (identical to computer copy)
c. [ ] Statement verifying identify of above copies

8. [ ] Assignment Papers (cover sheet & documents(s))

9. [ ] 37 CFR 3.73(b) Statement [ ] Power of Attorney
(*when there is an assignee*)

10. [ ] English Translation Document (*if applicable*)

11. [ ] Information Disclosure Statement (IDS)/PTO-1449 [ ] Copies of IDS Citations

12. [ ] Preliminary Amendment

13. [X] Return Receipt Postcard (MPEP 503)
(*Should be specifically itemized*)

14. [ ] Small Entity Statement(s) [ ] Statement filed in prior application, Status still proper and desired

15. [ ] Certified Copy of Priority Document(s)
(*if foreign priority is claimed*)

16. [ ] Other: .....

.....

.....

17. If a **CONTINUING APPLICATION**, *check appropriate box and supply the requisite information:*

[ ] Continuation [ ] Divisional [ ] Continuation-in-part (CIP) of prior application No.: ___/_____

16. CORRESPONDING ADDRESS

☐ *Customer Number of Bar Code Label* **021971** *(Insert Customer No. or Attach bar code label here)* *or* [X] *Correspondence address below*

NAME	*MARK A. HAYNES*				
ADDRESS	*WILSON SONSINI GOODRICH & ROSATI*				
	650 Page Mill Road				
CITY	*Palo Alto*	*STATE*	*California*	*ZIP CODE*	*94304-1050*
COUNTRY	*USA*	*TELEPHONE*	*(650) 493-9300*	*FAX*	*(650) 845-5000*

SUBMITTED BY
Typed or
Printed Name MARK A. HAYNES Reg. Number 30,846

Signature _____ Date October 16, 1998

DRH::ODMA\MHODMA\PALIB1;961652;1

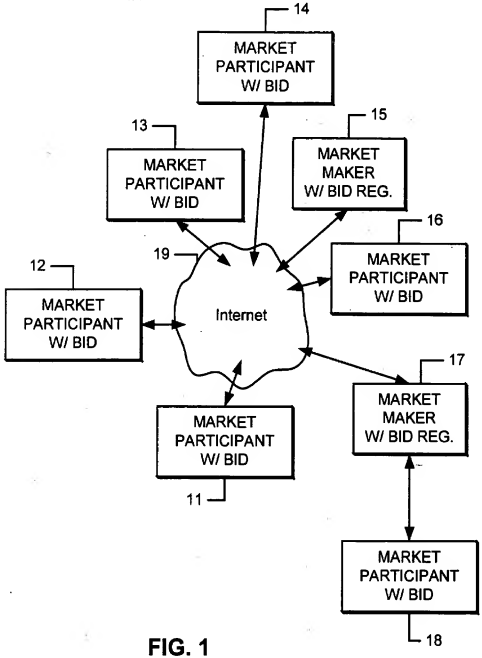
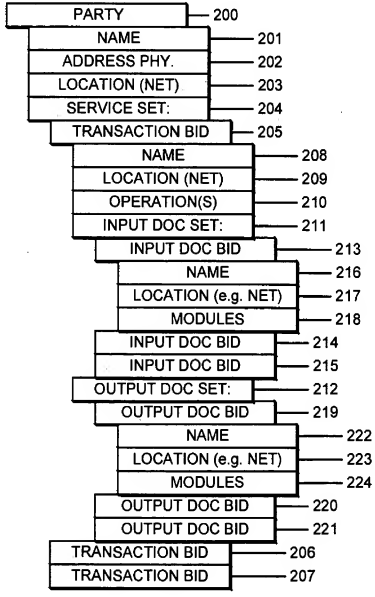


FIG. 1

FIG. 2



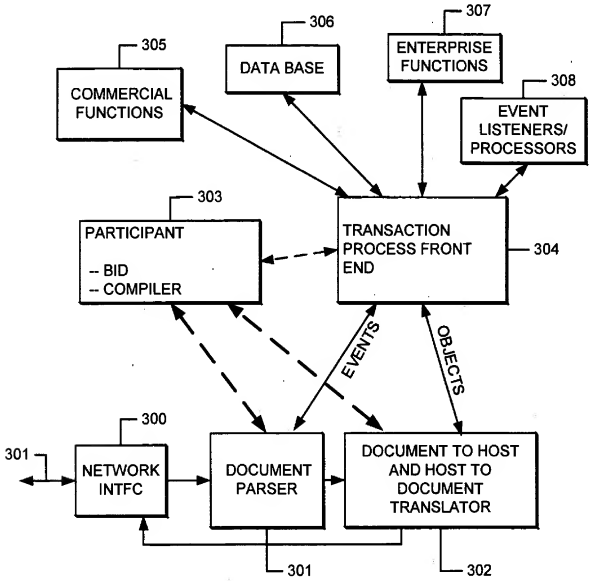


FIG. 3

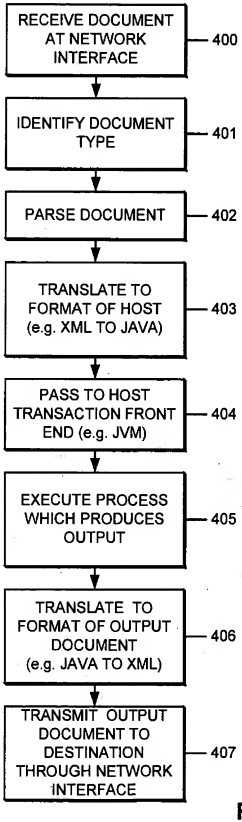


FIG. 4

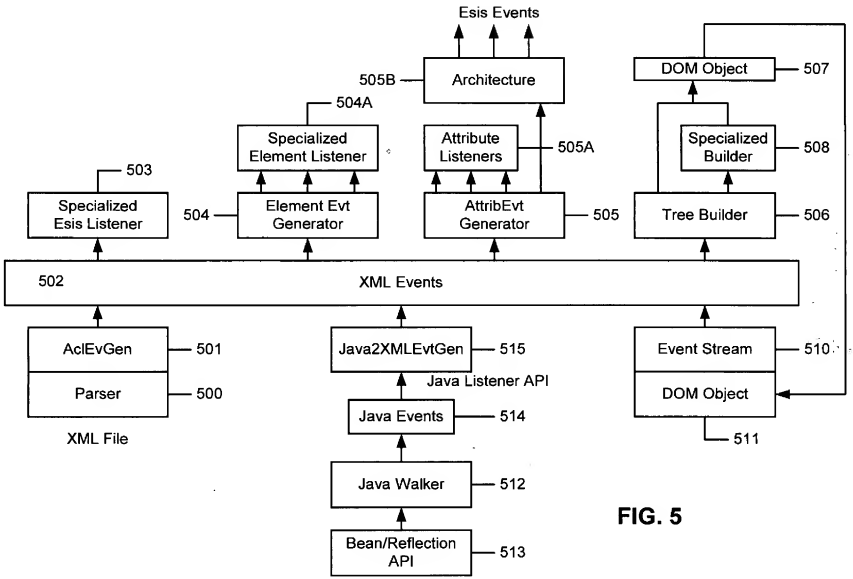
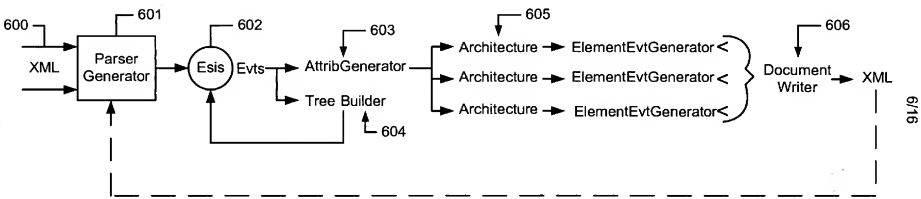


FIG. 5



6/16

FIG. 6

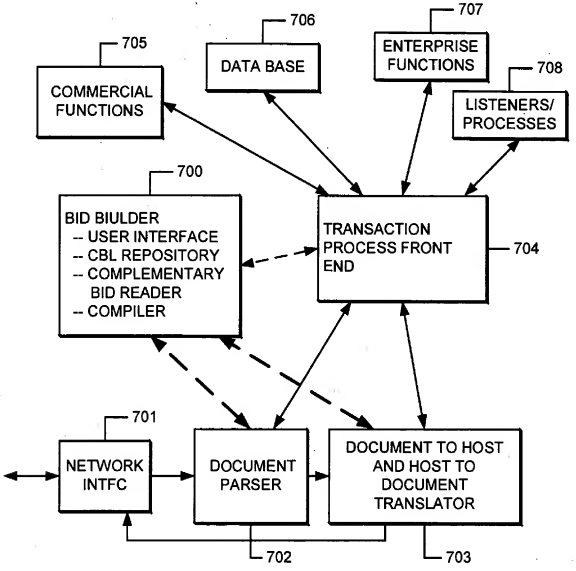


FIG. 7

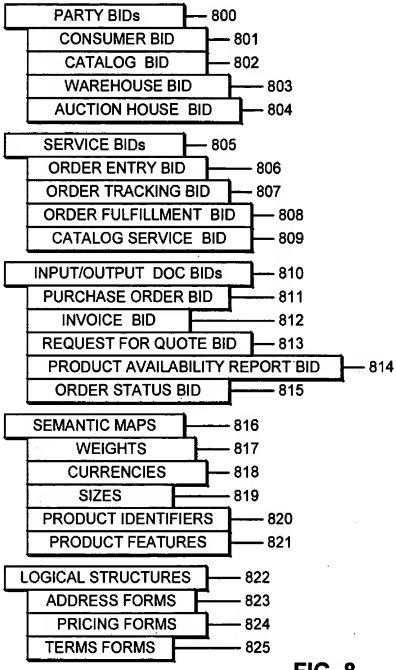


FIG. 8

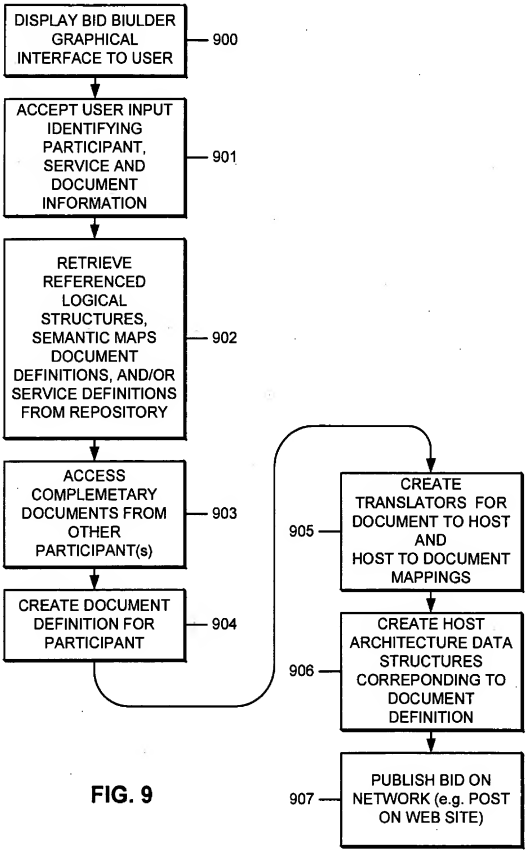
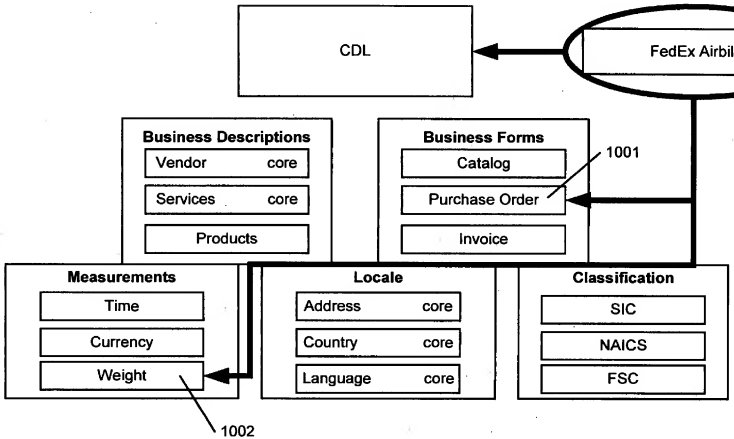


FIG. 9



10/16

FIG. 10

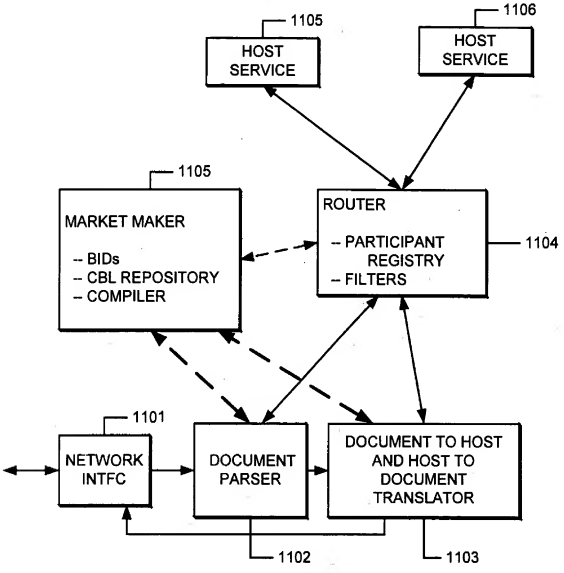


FIG. 11

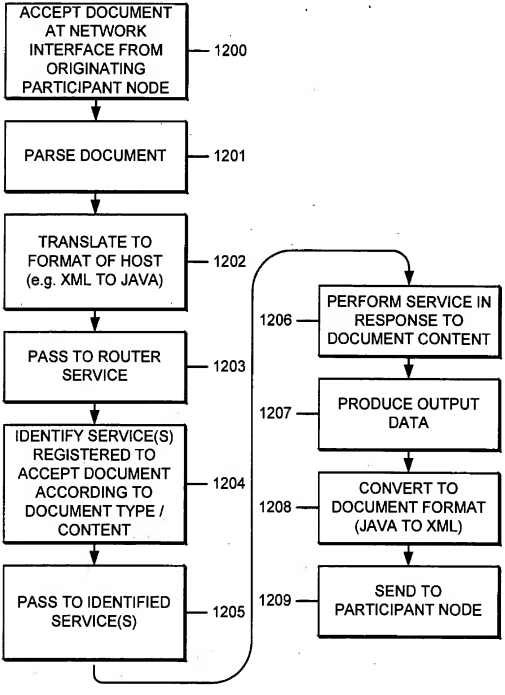


FIG. 12

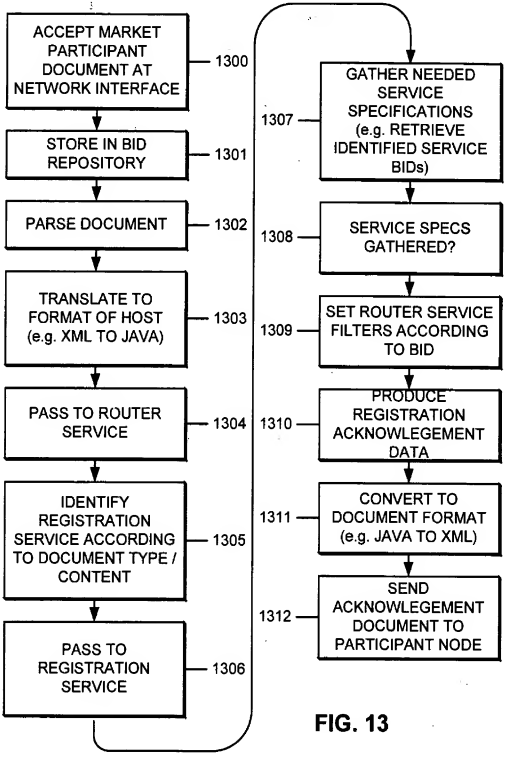


FIG. 13

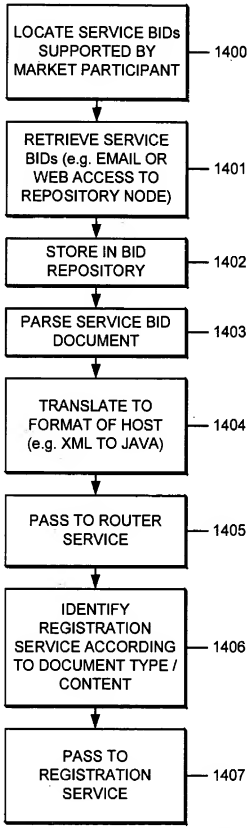


FIG. 14

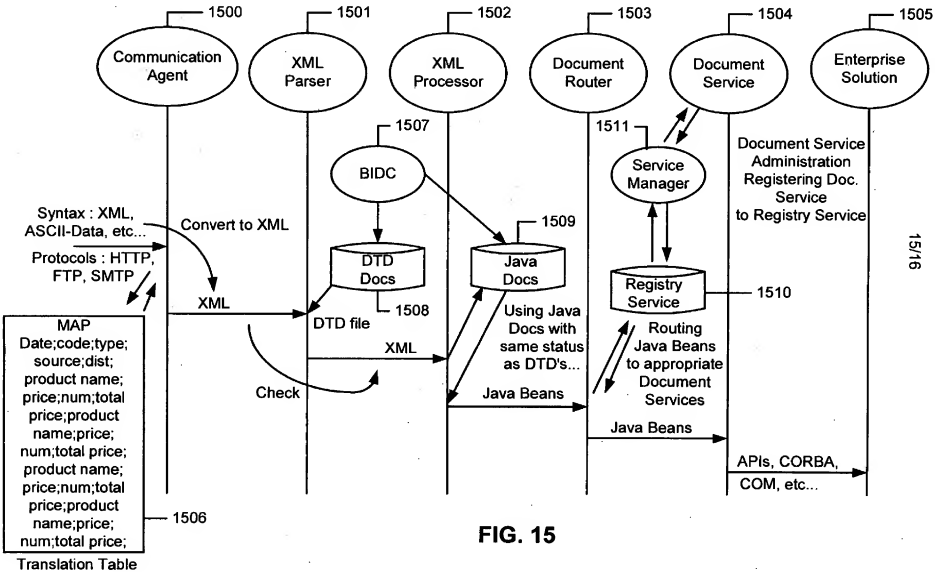


FIG. 15

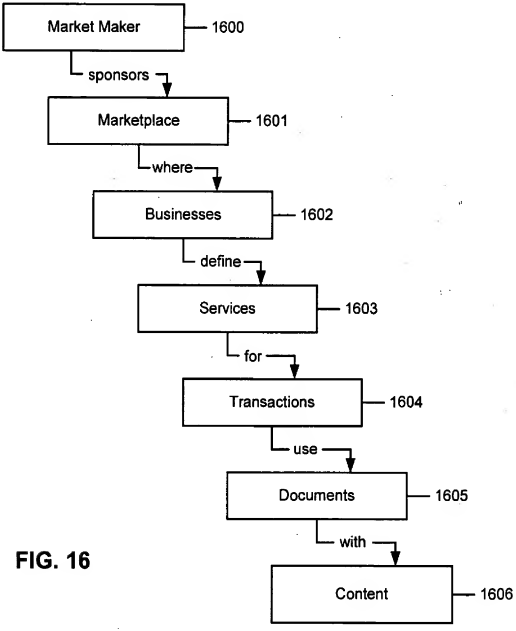


FIG. 16

DOCUMENTS FOR COMMERCE IN TRADING PARTNER NETWORKS AND INTERFACE DEFINITIONS BASED ON THE DOCUMENTS

Inventors: Bart Alan Meltzer
Terry Allen
Matthew Daniel Fuchs
Robert John Glushko
Murray Maloney

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application is related to co-pending U.S. patent application No. _____, filed on _____, the same day as the present application, and having the same inventors, entitled PARTICIPANT SERVER WHICH PROCESSES DOCUMENTS FOR COMMERCE IN TRADING PARTNER NETWORKS (19957-702); and

to co-pending U.S. patent application No. _____, filed on _____, the same day as the present application, and having the same inventors, entitled MARKET MAKERS USING DOCUMENTS FOR COMMERCE IN TRADING PARTNER NETWORKS (19957-703).

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates to systems and protocols supporting transactions among diverse clients coupled to a network; and more particularly to systems and protocols supporting commercial transactions among platforms having variant architectures.

Description of Related Art

The Internet and other communications networks provide avenues for communication among people and computer platforms which are being used for a wide variety of transactions, including commercial transactions in which participants buy and sell goods and services. Many efforts are underway to facilitate commercial transactions on the Internet. However, with many competing standards, in order to execute a transaction, the parties to the transaction must agree in advance on the protocols to be utilized, and often require custom integration of the platform architectures to support such transactions. Commercial processes internal to a particular node not compatible with agreed upon standards, may require substantial rework for integration with other nodes. Furthermore, as a company commits to one standard or the other, the company becomes locked-in to a given standardized group of transacting parties, to the exclusion of others.

A good overview of the challenges met by Internet commerce development is provided in Tenenbaum, et al., *"Eco System: An Internet Commerce Architecture"*, Computer, May 1997, pp. 48-55.

To open commercial transactions on the Internet, standardization of architectural frameworks is desired. Platforms developed to support such commercial frameworks include IBM Commerce Point, Microsoft Internet Commerce Framework, Netscape ONE (Open Network Environment), Oracle

NCA (Network Computing Architecture), and Sun/JAVASoft JECF (JAVA Electronic Commerce Framework).

In addition to these proprietary frameworks, programming techniques, such as common distributed object model based on CORBA IIOP (Common Object Request Broker Architecture Internet ORB Protocol), are being pursued. Use of the common distributed object model is intended to simplify the migration of enterprise systems to systems which can inter-operate at the business application level for electronic commerce. However, a consumer or business using one framework is unable to execute transactions on a different framework. This limits the growth of electronic commerce systems.

Companies implementing one framework will have an application programming interface API which is different than the API's supporting other frameworks. Thus, it is very difficult for companies to access each others business services, without requiring adoption of common business system interfaces. The development of such business system interfaces at the API level requires significant cooperation amongst the parties which is often impractical.

Accordingly, it is desirable to provide a framework which facilitates interaction amongst diverse platforms in a communication network. Such system should facilitate spontaneous commerce between trading partners without custom integration or prior agreement on industry wide standards. Further, such systems should encourage incremental path to business automation, to eliminate much of the time, cost and risks of traditional systems integration.

Overall, it is desirable to provide an electronic commerce system that replaces the closed trading partner networks based on proprietary standards with open markets.

SUMMARY OF THE INVENTION

The present invention offers an infrastructure for connecting businesses with customers, suppliers and trading partners. Under the infrastructure of the present invention, companies exchange information and services using self-defining, machine-readable documents, such as XML (Extensible Markup Language) based documents, that can be easily understood amongst the partners. Documents which describe the documents to be exchanged, called business interface definitions BIDs herein, are posted on the Internet, or otherwise communicated to members of the network. The business interface definitions tell potential trading partners the services the company offers and the documents to use when communicating with such services. Thus, a typical business interface definition allows a customer to place an order by submitting a purchase order, compliant with a document definition published in the BID of a party to receive the purchase order. A supplier is allowed to check availability by downloading an inventory status report compliant with a document definition published in the BID of a business system managing inventory data. Use of predefined, machine-readable business documents provides a more intuitive and flexible way to access enterprise applications.

A node in the commerce network establishes an interface for transactions according to the present invention that comprises a machine-readable specification of an interface, along with a machine-readable data structure that includes interpretation information for the machine-readable specification of the interface. The machine-readable specification of the interface includes a definition of an input document and a definition of an output document, that are accepted and produced by transaction processes for which the node acts as an interface. The definitions of the input and output documents comprise respective descriptions of sets of storage units and logical

structures for sets of storage units, such as according to a standard XML based document. The machine-readable data structure that includes interpretation information according to various aspects of the invention includes data type specifications (e.g. string, array, etc.) for logical structures in the definitions of the input and output documents, content models (e.g. lists of possible values) for logical structures and/or data structures that map predefined sets of storage units for a particular logic structure to respective entries in a list in order to provide a semantic definition of logical structures (e.g. mapping codes to product names).

According to other aspects of the invention, the interface includes a repository in memory accessible by at least one node in the network that stores a library of logic structures and interpretation information for the logic structures. The repository can be extended to include a library of definitions of input and output documents, a library of specifications of interfaces, and a library of specifications for participant interface nodes in the network.

Thus, a participant in the transaction framework of the present invention executes transactions amongst nodes in a network that includes a plurality of nodes executing processes involved in the transactions. The method includes storing a machine-readable specification of an interface for a transaction, the specification includes a definition of an input document and a definition of an output document. The definition of the input and output documents comprise respective descriptions of sets of storage units and logical structures for the sets of storage units. In a preferred system, the definitions are expressed in a manner compliant with a standard XML document type definition DTD, extended by semantic mapping, content modeling and data typing of some elements. The participant in the transaction receives data comprising a document through a communication network. The participant parses the document according to the specification stored for a transaction to identify an input document for the

transaction. After parsing the document, at least a portion of the input document is provided in a machine-readable format to a transaction process which produces an output. An output document is formed comprising the output of the transaction process, based on the definition of an output document in the stored specification. The output document is transmitted on the communication network, typically back to the source of the input document, or elsewhere as suits the needs of a particular type of transaction.

Thus the business interface definition bridges the gap between the documents specified for example according to XML and the programs which execute on the front end of the transaction processing services at particular nodes. Such front ends are implemented for example by JAVA virtual machines, or by other common architectures providing for interconnection of systems across a network. The business interface definition provides a technique by which a transaction protocol is programmed using the business interface definition document. The program for the protocol of the transaction is established by a detailed formal specification of a document type.

The machine-readable specification of the interface of the transaction is made accessible to other platforms in the network. Participant platforms include resources to design input documents and output documents according to the transaction interface specified at a complementary node. Thus, participant nodes include resources to access the definition of an input document for the complementary interface and a definition of an output document for the complementary interface. The stored specification for the accessing participant node is established by including at least part of the definition of the output document of the complementary interface in the definition of the input document of the interface stored in the specification.

The process of establishing the stored specification of an interface according to another aspect of the invention includes accessing elements of the

machine-readable specification from a repository. The repository stores a library of logic structures, content models, and schematic maps for logic structures, and definition of documents that comprise logic structures used to build interface description. A repository accessible in the network facilitates the building of interface descriptions which can be easily shared. Any differences between the input document created for a particular process and the output document expected as a return by a complementary process can be easily negotiated by communication on the network and agreeing on common logic structures to express particular information.

The machine-readable specification of an interface of a transaction according to one aspect of the invention includes a document compliant with a definition of an interface document that is shared amongst members of the network. A definition of the interface document includes logic structures for storing an identifier of a particular transaction and at least one of definitions and references to definitions of input and output documents for the particular transaction. That is, the interface description for a particular service may actually encompass a definition of the input and output documents. Alternatively, it may include pointers to a location in the repository, or elsewhere in the network, of such definitions.

According to another alternative of the invention, the machine-readable specification includes a business interface definition BID document compliant with a definition of an interface document that includes logical structures for storing an identifier of the interface, and for storing at least one of specifications and references to specifications of a set of one or more transactions supported by the interface. For each supported transaction, the document includes at least one of definitions and references to definitions of input and output documents for the particular transaction.

According to another aspect of the invention, the storage units defined in the definitions of the input and output document comprise parsed data including character data encoding text characters, and mark-up data identifying sets of storage units according to the logical structures of the input and output documents. According to another aspect of the invention, at least one of the sets of storage units encodes the plurality of text characters providing a natural language word. This facilitates the use of the definitions of input and output documents by human readers and developers of such documents.

According to another aspect of the invention, the specification of the input and output documents includes interpretation information for at least one of the sets of storage units identified by the logical structure. The interpretation information in one example encodes definitions for sets of parsed characters. In another example, the interpretation information provides for content model specifications, such as requiring a specific logic structure to carry a member of a list of codes mapped to product descriptions in a catalog. In some systems, the storage units in a logic structure of a document may include sets of unparsed data, as suits the needs of a particular implementation.

According to yet another aspect of the invention, the transaction process in a particular platform has a transaction processing architecture which is one of a plurality of variant transaction processing architectures. Thus the participant node includes resources for translating at least a portion of the input document into a format readable according to the variant transaction processing architecture of the transaction process utilizing the information. The elements of the document definition are translated into programming objects that include variables and methods according to the variant transaction processing architectures of the transaction process. For a participant having a JAVA virtual machine acting as a transaction process front end, particular fields in the documents are translated into JAVA objects, including the data as well as get

and set functions associated with a JAVA object. Other transaction processing architectures supportable according to the present invention include processes compliant with an interface description language in the sense of Common Object Request Broker Architecture CORBA, Component Object Model COM,On-Line Transaction Processing OLTP, and Electronic Data Interchange EDI.

According to other aspects of the invention, a repository is provided that stores document types for use in the plurality of transactions. The machine-readable specification for a particular transaction defines at least one of the input and output documents by reference to a document type in the repository. According to another aspect, the document type included in the repository include a document type for identifying participants in the network. Such document type providing structures for identifying a participant, specifying the services supported by the participant, and specifying the input and output documents for each of such services.

In addition to the methods described above, the present invention provides an apparatus for managing transactions among nodes that includes a network interface, memory for storing data and programs of instructions including a machine-readable specification of an interface for a transaction as described above, and a data processor that is coupled with the memory and the network interface. The programs of instructions stored in the apparatus include logic to execute the processes described above for a participant in the transactions.

The present invention further provides an apparatus for establishing participant interfaces for transactions executed on a system that include a network interface and a data processing resources that execute transaction processes according to a transaction processing architecture. The apparatus includes programs of instructions that are executable by the system and stored

on a medium accessible by the system that provide tools to build a definition of a participant interface for a participant in a particular transaction. The definition of a participant interface includes a definition of an input document and a definition of an output document. The definitions of the input and output documents comprise respective machine-readable descriptions of sets of storage units in logical structures for the sets of storage units, which may be compliant in one aspect of the invention with XML document type definitions.

The apparatus for establishing participant interfaces according to this aspect of the invention also includes programs of instructions stored on a medium accessible by the data processing system and responsive to the definitions of input and output documents to compile data structures corresponding to the sets of storage units and logical structures of the input and output documents that are compliant with the transaction processing architecture, to compile instructions executable by the system to translate the input document to the corresponding data structures, and to compile instructions executable by the system to translate output of the transaction processes into sets of storage units and logical structures of the output document.

The tools to build a definition of a participant interface in a preferred system include instructions executable by the system to access elements of the definition from complementary nodes and/or from a repository that stores a library of logical structures and interpretation information for logical structures used to build interface descriptions. According to various aspects of the invention, the repository includes not only a library of logical structures but definitions of documents that comprise logical structures, and formats for specifying participant interfaces. According to this aspect of the invention, tools are provided for building specifications of business interfaces according to the techniques described above in connection with the description of the participant nodes. The tools for building interfaces and the tools for compiling

the interfaces into resources needed for communication with transaction processing architectures according to this aspect of the invention, are implemented in participant nodes in the preferred system, and utilized for the development and optimization of the interface descriptions as use of the network grows based on interface descriptions that define input and output documents.

Accordingly, another aspect of the invention provides an apparatus that includes memory and a data processor that executes instructions stored in the memory that include tools to build a definition of a participant interface and a compiler performing the functions described above.

According to yet another aspect of the invention, the use of the participant interface descriptions enables the operation of a market maker node. At such a node, a method for managing transactions is provided that comprises storing machine-readable specifications of a plurality of participant interfaces which identify transactions supported by the interfaces, and the respective input and output documents of such transactions. As described above, the definitions of the input and output documents comprise respective descriptions of sets of storage units and logical structures for the sets of storage units, such as according to the XML standard. In addition, the definitions of the transactions and the definitions of the participant interfaces all comprise documents specified according to a technique compliant with XML or other standardized document expression language. At such market maker node, data comprising a document is received over a communication network. The document is parsed according to the specifications to identify an input document in one or more transactions which accept the identified input document. At least a portion of the input document is provided in a machine-readable format to a transaction process associated with the one or more identified transactions. The step of providing at least a portion of the input document to a transaction process

includes executing a routing process according to a processing architecture at the market maker node. The routing process in one aspect of the invention is executed according to a particular processing architecture, and at least a portion of the incoming document is translated into the format of the processing architecture of the routing process. The translating according to the preferred aspect includes producing programming objects that include variables and methods according to the processing architecture of the routing process.

According to another aspect of the invention, the market maker node also supports the repository structure. Thus, a process is included at the market maker node for allowing access by participants in the network to a repository stored at the market maker node. As described above, the repository includes definitions of logic structures, interpretation information, and document definitions for use by the participant nodes to build transaction interface documents and includes instances of business interface definitions that identify the participants, and the transactions executed by the respective participants.

The routing process includes according to various alternatives the translating of the input document into the variant processing architecture of the processes to which the document is to be routed, or routing the input document in its original document format across the network to a remote processing node, or to combinations of such processes. In alternatives, the routing process may also include techniques for transforming an input document defined according to one input document definition into a different document defined according to a different document specification for a process which has registered to watch for the input document.

Also, the market maker node is provided according to the present invention as apparatus that includes a network interface, memory storing data and programs of instructions including the specifications of the participant interfaces, and a data processor. The logic is provided with the data processor

in the form of programs of instructions or otherwise to perform the market maker process as discussed above.

Accordingly, the present invention provides a foundation for electronic commerce based on the sharing of specifications of input and output documents. Documents provide an intuitive and flexible way to access business services, much simpler to implement than programming APIs. It is much easier to interconnect companies in terms of documents that are exchanged, on which such companies already largely agree, than in terms of business system interfaces which invariably differ. In addition, such documents are specified in a human readable format in the preferred embodiment. According to the present invention the business interfaces are specified in documents, such as XML documents that are readily interpretable by humans as well as by computers.

Utilization of the document based transaction architecture of the present invention involves the use of a parser which operates in basically the same way for any source of documents, eliminating much of the need for custom programs to extract and integrate information from each participant in the network. Thus, integrating enterprise information from accounting, purchasing, manufacturing, shipping and other functions can be accomplished by first converting each source to a document having an architecture according to the present invention, and then processing the parsed data stream. Each node in the system that participates in the market only needs to know about the format of its internal systems, as well as the format of the documents being specified for interchange according to the transaction. Thus, there is no need to be able to produce the native format of every other node which might want to participate in the electronic commerce network.

For complete business integration, the present invention provides a repository of standardized logical structures, like XML elements, attributes or meta data, establishing a semantic language for particular commerce

communities, a means for mapping between different meta data descriptions, and a server for processing the documents and invoking appropriate applications and services. The basic building blocks of a network according to the present invention include the business interface definitions which tell potential trading partners what online services a company offers and which documents to use to invoke the services; and servers which provide the bridge to bind together the set of internal and external business services to create a trading community. The server operates to parse the incoming documents and invoke the appropriate services. Also the server according to the present invention handles the translation tasks from the format of the documents being received and transmitted, to and from the formats of the respective host systems. Thus, trading partners need only agree on the structure, content and sequencing of the business documents exchanged, and not on the details of application programmer interfaces. How a document is processed and the actions which result from receipt of a document are strictly up to the businesses providing the services. This elevates integration from the system level to the business level. It enables the business to present a clean and stable interface to its partners despite changes in its internal technology implementation, organization or processes.

The whole process of building business interface definitions and enabling servers to manage commerce according to such descriptions is facilitated by a common business library, or repository, of information models for generic business concepts including business description primitives like companies, services and products, business forms like catalogs, purchase orders and invoices, and standard measurements, including time and date, location and classification of goods.

Other aspects of the present invention can be seen upon review of the figures, the detailed description, and the claims which follow.

BRIEF DESCRIPTION OF THE FIGURES

Fig. 1 is a simplified diagram of an electronic commerce network including business interface definitions BIDs according to the present invention.

Fig. 2 is a simplified diagram of a business interface definition document according to the present invention.

Fig. 3 is a conceptual block diagram of a server for a participant node in the network of the present invention.

Fig. 4 is a flow chart illustrating the processing of a received document at a participant node according to the present invention.

Fig. 5 is a block diagram of a parser and transaction process front end for an XML based system.

Fig. 6 is a conceptual diagram of the flow of a parse function.

Fig. 7 is a simplified diagram of the resources at a server used for building a business interface definition according to the present invention.

Fig. 8 is a simplified diagram of a repository according to the present invention for use for building business interface definitions.

Fig. 9 is a flow chart illustrating the processes of building a business interface definition according to the present invention.

Fig. 10 provides a heuristic view of a repository according to the present invention.

Fig. 11 is a simplified diagram of the resources at a server providing the market maker function for the network of the present invention based on business interface definitions.

Fig. 12 is a flow chart for the market maker node processing of a received document.

Fig. 13 is a flow chart illustrating the process of registering participants at a market maker node according to the present invention.

Fig. 14 is a flow chart illustrating the process of providing service specifications at a market maker node according to the process of Fig. 9.

Fig. 15 is a diagram illustrating the sequence of operation at a participant or market maker node according to the present invention.

Fig. 16 is a conceptual diagram of the elements of a commercial network based on BIDs, according to the present invention.

DETAILED DESCRIPTION

A detailed description of the present invention is provided with respect to the figures, in which Fig. 1 illustrates a network of market participants and market makers based on the use of business interface definitions, and supporting the trading of input and output documents specified according to such interface descriptions. The network includes a plurality of nodes 11-18 which are interconnected through a communication network such as the Internet 19, or other telecommunications or data communications network. Each of the nodes 11-19 consists of a computer, such as a portable computer, a desktop personal computer, a workstation, a network of systems, or other data processing resources. The nodes include memory for storing the business interface definition, processors that execute transaction processes supporting commercial transactions with other nodes in the network, and computer programs which are executed by the processors in support of such services. In addition each of the nodes includes a network interface for providing for communication across the Internet 19, or the other communication network.

In the environment of Fig. 1, nodes 11, 12, 13, 14, 16 and 18 are designated market participants. Market participants include resources for consumers or suppliers of goods or services to be traded according to commercial transactions established according to the present invention.

In this example, nodes 15 and 17 are market maker nodes. The market maker nodes include resources for registering business interface definitions, called a BID registry. Participants are able to send documents to a market maker node, at which the document is identified and routed to an appropriate participant which has registered to receive such documents as input. The market maker also facilitates the commercial network by maintaining a repository of standard forms making up a common business library for use in building business interface definitions.

In this example, the market participant 18 is connected directly to the market maker 17, rather than through the Internet 19. This connection directly to the market maker illustrates that the configuration of the networks supporting commercial transactions can be very diverse, incorporating public networks such as the Internet 19, and private connections such as a local area network or a Point-to-Point connection as illustrated between nodes 17 and 18. Actual communication networks are quite diverse and suitable for use to establish commercial transaction networks according to the present invention.

Fig. 2 is a heuristic diagram of nested structures in a business interface definition BID which is established for market participants in the network according to the present invention. The business interface definition illustrated in Fig. 2 is a data structure that consists of logic structures and storage units arranged according to a formal definition of a document structure, such as a XML document type definition DTD. The structure of Fig. 2 includes a first logic structure 200 for identifying a party. Associated with the logic structure 200 are nested logic structures for carrying the name 201, the physical address 202, the network address or location 203, and a set of transactions for a service 204. For each transaction in the service set, an interface definition is provided, including the transaction BID 205, the transaction BID 206, and the transaction BID 207. Within each transaction BID, such as transaction BID 205, logical

structures are provided for including a name 208, a location on the network at which the service is performed 209, the operations performed by the service 210, and a set of input documents indicated by the tag 211. Also, the service BID 205 includes a set of output documents indicated by the tag 212. The set of input documents 211 includes a business interface definition for each input document for which the services are designed to respond, including input document business interface definitions 213, 214, and 215. Each business interface definition for an input document includes a name 216, a location on the network at which a description of the document can be found 217, and the modules to be carried in the document as indicated by the field 218. In a similar manner, the output document set 212 includes interface definitions for output documents including the output document BID 219, output document BID 220, and output document BID 221. For each output document BID, a name 222, a location on the network or elsewhere 223, and the modules of the document 224 are specified. The business interface definition for the participant as illustrated in Fig. 2 includes actual definitions of a logic structures to be utilized for the input and output documents of the respective services, or pointers or other references to locations at which these definitions can be found.

In the preferred system, the document of Fig. 2 is specified in an XML document type definition DTD, although other document definition architectures could be used, and includes interpretation information for the logical structures used in interpretation of instances of the documents. In addition, each of the transaction BIDs, input document BIDs and output document BIDs are specified according to an XML document type definitions. The XML type document is an example of a system based on parsed data that includes mark-up data and character data. Mark-up data identifies logical structures within the document and sets of character data identify the content of the logical structures. In addition, unparsed data can be carried in the document

for a variety of purposes. See for example the specification of the *Extensible Mark-up Language XML 1.0* REC-XML-19980210 published by the WC3 XML Working Group at WWW.W3.ORG/TR/1998/REC-XML-19980210.

Thus in an exemplary system, participant nodes in the network establish virtual enterprises by interconnecting business systems and services with XML encoded documents that businesses accept and generate. For example, the business interface definition of a particular service establishes that if a document matching the BID of a request for a catalog entry is received, then a document matching a BID of a catalog entry will be returned. Also, if a document matching the BID of a purchase order is received, and it is acceptable to the receiving terminal, a document matching the BID of an invoice will be returned. The nodes in the network process the XML documents before they enter the local business system, which is established according to the variant transaction processing architecture of any given system in the network. Thus, the system unpacks sets of related documents, such as MIME-encoded sets of XML documents, parses them to create a stream of "mark-up messages". The messages are routed to the appropriate applications and services using for example an event listener model like that described below.

The documents exchanged between business services are encoded using an XML language built from a repository of building blocks (a common business language) from which more complex document definitions may be created. The repository stores modules of interpretation information that are focused on the functions and information common to business domains, including business description primitives like companies, services and products; business forms like catalogs, purchase orders and invoices; standard measurements, like time, date, location; classification codes and the like providing interpretation information for logical structures in the XML documents.

The business interface definition is a higher level document that acts as a schema used for designing interfaces that trade documents according to the present invention. Thus the business interface definition bridges the gap between the documents specified according to XML and the programs which execute on the front end of the transaction processing services at particular nodes. Such front ends are implemented by JAVA virtual machines, or by other common architectures providing for interconnection of systems across a network. Thus, the business interface definition provides a technique by which a transaction protocol is programmed using the business interface definition document. The program for the protocol of the transaction is established by a detailed formal specification of a document type.

An example business interface definition BID based on a market participant document which conforms to an XML format is provided below. The market participant DTD groups business information about market participants, associating contact and address information with a description of services and financial information. This business information is composed of names, codes, addresses, a dedicated taxonomic mechanism for describing business organization, and a pointer to terms of business. In addition, the services identified by the market participant DTD will specify the input and output documents which that participant is expected respond to and produce. Thus, documents which define schema using an exemplary common business language for a market participant DTD, a service DTD, and a transaction document DTD specified in XML with explanatory comments follow:

Market Participant Sample

```
<!DOCTYPE SCHEMA SYSTEM "bidl.dtd">
<SCHEMA>
<H1>Market Participant Sample BID</H1>
<META
WHO.OWNS="Veo Systems"          WHO.COPYRIGHT="Veo Systems"
WHEN.COPYRIGHT="1998"           DESCRIPTION="Sample BID"
WHO.CREATED="*"                 WHEN.CREATED="*"
WHAT.VERSION="*"                WHO.MODIFIED="*"
WHEN.MODIFIED="*"               WHEN.EFFECTIVE="*"
WHEN.EXPIRES="*"                WHO.EFFECTIVE="*"
WHO.EXPIRES="*">
</META>

<PROLOG>
<XMLDECL STANDALONE="no"></XMLDECL>
<DOCTYPE NAME="market.participant">
<SYSTEM>markpart.dtd</SYSTEM></DOCTYPE>
</PROLOG>

<DTD NAME="markpart.dtd">
<H2>Market Participant</H2>
<H3>Market Participant</H3>
<ELEMENTTYPE NAME="market.participant">
<EXPLAIN><TITLE>A Market Participant</TITLE>
<SYNOPSIS>A business or person and its service interfaces.</SYNOPSIS>
<P>A market participant is a document definition that is created to describe a business and at
least one person with an email address, and it presents a set of pointers to service interfaces
located on the network. In this example, the pointers have been resolved and the complete BID
is presented here.</P></EXPLAIN>
<MODEL><CHOICE>
<ELEMENT NAME="business"></ELEMENT>
<ELEMENT NAME="person"></ELEMENT>
</CHOICE></MODEL></ELEMENTTYPE>

<H3>Party Prototype</H3>
<PROTOTYPE NAME="party">
<EXPLAIN><TITLE>The Party Prototype</TITLE></EXPLAIN>
<MODEL><SEQUENCE>
<ELEMENT NAME="party.name" OCCURS="+"></ELEMENT>
<ELEMENT NAME="address.set"></ELEMENT>
</SEQUENCE></MODEL>
</PROTOTYPE>

<H3>Party Types</H3>
<ELEMENTTYPE NAME="business">
<EXPLAIN><TITLE>A Business</TITLE>
<SYNOPSIS>A business (party) with a business number attribute.</SYNOPSIS>
```

```
<P>This element inherits the content model of the party prototype and adds a business number
attribute, which serves as a key for database lookup. The business number may be used as a
cross-reference to/from customer id, credit limits, contacts lists, etc.</P></EXPLAIN>
<EXTENDS HREF="party">
<ATTDEF NAME="business.number"><REQUIRED></REQUIRED></ATTDEF>
</EXTENDS>
</ELEMENTTYPE>

<H3>Person Name</H3>
<ELEMENTTYPE NAME="person">
<EXPLAIN><TITLE>A Person</TITLE></EXPLAIN>
<EXTENDS HREF="party">
<ATTDEF NAME="SSN"><IMPLIED></IMPLIED></ATTDEF>
</EXTENDS>
</ELEMENTTYPE>

<H3>Party Name</H3>
<ELEMENTTYPE NAME="party.name">
<EXPLAIN><TITLE>A Party's Name</TITLE>
<SYNOPSIS>A party's name in a string of character.</SYNOPSIS></EXPLAIN>
<MODEL><STRING></STRING></MODEL>
</ELEMENTTYPE>

<H3>Address Set</H3>
<ELEMENTTYPE NAME="address.set">
<MODEL><SEQUENCE>
<ELEMENT NAME="address.physical"></ELEMENT>
<ELEMENT NAME="telephone" OCCURS="*"></ELEMENT>
<ELEMENT NAME="fax" OCCURS="*"></ELEMENT>
<ELEMENT NAME="email" OCCURS="*"></ELEMENT>
<ELEMENT NAME="internet" OCCURS="*"></ELEMENT>
</SEQUENCE></MODEL>
</ELEMENTTYPE>

<H3>Physical Address</H3>
<ELEMENTTYPE NAME="address.physical">
<EXPLAIN><TITLE>Physical Address</TITLE>
<SYNOPSIS>The street address, city, state, and zip code.</SYNOPSIS></EXPLAIN>
<MODEL><SEQUENCE>
<ELEMENT NAME="street"></ELEMENT>
<ELEMENT NAME="city"></ELEMENT>
<ELEMENT NAME="state"></ELEMENT>
<ELEMENT NAME="postcode" OCCURS="?"></ELEMENT>
<ELEMENT NAME="country"></ELEMENT>
</SEQUENCE></MODEL>
</ELEMENTTYPE>

<H3>Street</H3>
<ELEMENTTYPE NAME="street">
<EXPLAIN><TITLE>Street Address</TITLE>
```

```
<SYNOPSIS>Street or postal address.</SYNOPSIS></EXPLAIN>
<MODEL><STRING></STRING></MODEL>
</ELEMENTTYPE>

<H3>City</H3>
<ELEMENTTYPE NAME="city">
<EXPLAIN><TITLE>City Name or Code</TITLE>
<P>The city name or code is a string that contains sufficient information to identify a city
within a designated state.</P>
</EXPLAIN>
<MODEL><STRING></STRING></MODEL>
</ELEMENTTYPE>

<H3>State</H3>
<ELEMENTTYPE NAME="state">
<EXPLAIN><TITLE>State, Province or Prefecture Name or Code</TITLE>
<P>The state name or code contains sufficient information to identify a state within a
designated country.</P></EXPLAIN>
<MODEL><STRING DATATYPE="COUNTRY.US.SUBENTITY"></STRING></MODEL>
</ELEMENTTYPE>

<H3>Postal Code</H3>
<ELEMENTTYPE NAME="postcode">
<EXPLAIN><TITLE>Postal Code</TITLE>
<P>A postal code is an alphanumeric code, designated by an appropriate postal authority, that
is used to identify a location or region within the jurisdiction of that postal authority. Postal
authorities include designated national postal authorities.</P></EXPLAIN>
<MODEL><STRING DATATYPE="string"></STRING></MODEL>
</ELEMENTTYPE>

<H3>Country</H3>
<ELEMENTTYPE NAME="country">
<EXPLAIN><TITLE>Country Code</TITLE>
<P>A country code is a two-letter code, designated by ISO, that is used to uniquely identify a
country.</P></EXPLAIN>
<MODEL><STRING DATATYPE="country"></STRING></MODEL>
</ELEMENTTYPE>

<H3>Network Addresses</H3>
<ELEMENTTYPE NAME="telephone">
<EXPLAIN><TITLE>Telephone Number</TITLE>
<P>A telephone number is a string of alphanumerics and punctuation that uniquely identifies a
telephone service terminal, including extension number.</P></EXPLAIN>
<MODEL><STRING></STRING></MODEL>
</ELEMENTTYPE>

<H3>Fax</H3>
<ELEMENTTYPE NAME="fax">
<EXPLAIN><TITLE>Fax Number</TITLE>
```

```
<P>A fax number is a string of alphanumerics and punctuation that uniquely identifies a fax
service terminal.</P>
</EXPLAIN>
<MODEL><STRING></STRING></MODEL>
</ELEMENTTYPE>

<H3>Email</H3>
<ELEMENTTYPE NAME="email">
<EXPLAIN><TITLE>Email Address</TITLE>
<P>An email address is a datatype-constrained string that uniquely identifies a mailbox on a
server.</P></EXPLAIN>
<MODEL><STRING DATATYPE="email"></STRING></MODEL>
</ELEMENTTYPE>

<H3>Internet Address</H3>
<ELEMENTTYPE NAME="internet">
<EXPLAIN><TITLE>Internet Address</TITLE>
<P>An Internet address is a datatype-constrained string that uniquely identifies a resource on
the Internet by means of a URL.</P></EXPLAIN>
<MODEL><STRING DATATYPE="url"></STRING></MODEL>
</ELEMENTTYPE>

</DTD>
</SCHEMA>
```

Service Description Sample

```
<!DOCTYPE schema SYSTEM "bidl.dtd">
<SCHEMA>
<H1>Service Description Sample BID</H1>
<META
WHO.OWNS="Veo Systems"          WHO.COPYRIGHT="Veo Systems"
WHEN.COPYRIGHT="1998"           DESCRIPTION="Sample BID"
WHO.CREATED="*"                 WHEN.CREATED="*"
WHAT.VERSION="*"                WHO.MODIFIED="*"
WHEN.MODIFIED="*"               WHEN.EFFECTIVE="*"
WHEN.EXPIRES="*"                WHO.EFFECTIVE="*"
WHO.EXPIRES="*">
</META>

<PROLOG>
<XMLDECL STANDALONE="no"></XMLDECL>
<DOCTYPE NAME="service">
<SYSTEM>service.dtd</SYSTEM></DOCTYPE>
</PROLOG>

<DTD NAME="service.dtd">
<H2>Services</H2>

<H3>Includes</H3>
```

```
<!-- INCLUDE><SYSTEM>comments.bim</SYSTEM></INCLUDE -->

<H3>Service Set</H3>
<ELEMENTTYPE NAME="service.set">
<EXPLAIN><TITLE>Service Set</TITLE>
<SYNOPSIS>A set of services.</SYNOPSIS></EXPLAIN>
<MODEL>
<ELEMENT NAME="service" OCCURS="+"></ELEMENT>
</MODEL></ELEMENTTYPE>

<H3>Services Prototype</H3>
<PROTOTYPE NAME="prototype.service">
<EXPLAIN><TITLE>Service</TITLE></EXPLAIN>
<MODEL><SEQUENCE>
<ELEMENT NAME="service.name"></ELEMENT>
<ELEMENT NAME="service.terms" OCCURS="+"></ELEMENT>
<ELEMENT NAME="service.location" OCCURS="+"></ELEMENT>
<ELEMENT NAME="service.operation" OCCURS="+"></ELEMENT>
</SEQUENCE></MODEL>
<!-- ATTGROUP><IMPLEMENTS
HREF="common.attrib"></IMPLEMENTS></ATTGROUP -->
</PROTOTYPE>

<H3>Service</H3>
<INTRO><P>A service is an addressable network resource that provides interfaces to specific
operations by way of input and output documents.</P></INTRO>
<ELEMENTTYPE NAME="service">
<EXPLAIN><TITLE>Service</TITLE>
<P>A service is defined in terms of its name, the location(s) at which the service is available,
and the operation(s) that the service performs.</P></EXPLAIN>
<MODEL><SEQUENCE>
<ELEMENT NAME="service.name"></ELEMENT>
<ELEMENT NAME="service.location"></ELEMENT>
<ELEMENT NAME="service.operation" OCCURS="+"></ELEMENT>
<ELEMENT NAME="service.terms"></ELEMENT>
</SEQUENCE></MODEL>
</ELEMENTTYPE>

<H3>Service Name</H3>
<ELEMENTTYPE NAME="service.name">
<EXPLAIN><TITLE>Service Name</TITLE>
<P>The service name is a human-readable string that ascribes a moniker for a service. It may be
employed is user interfaces and documentation, or for other purposes.</P></EXPLAIN>
<MODEL><STRING></STRING></MODEL>
</ELEMENTTYPE>

<H3>Service Location</H3>
<ELEMENTTYPE NAME="service.location">
<EXPLAIN><TITLE>Service Location</TITLE>
<SYNOPSIS>A URI of a service.</SYNOPSIS>
```

```
<P>A service location is a datatype-constrained string that locates a service on the Internet by
means of a URI.</P></EXPLAIN>
<MODEL><STRING DATATYPE="url"></STRING></MODEL>
</ELEMENTTYPE>

<H3>Service Operations</H3>
<INTRO><P>A service operation consists of a name, location and its interface, as identified by
the type of input document that the service operation accepts and by the type of document that it
will return as a result.</P></INTRO>
<ELEMENTTYPE NAME="service.operation">
<EXPLAIN><TITLE>Service Operations</TITLE>
<P>A service operation must have a name, a location, and at least one document type as an
input, with one or more possible document types returned as a result of the operation.</P>
</EXPLAIN>
<MODEL><SEQUENCE>
<ELEMENT NAME="service.operation.name"></ELEMENT>
<ELEMENT NAME="service.operation.location"></ELEMENT>
<ELEMENT NAME="service.operation.input"></ELEMENT>
<ELEMENT NAME="service.operation.output"></ELEMENT>
</SEQUENCE></MODEL>
</ELEMENTTYPE>

<H3>Service Operation Name</H3>
<ELEMENTTYPE NAME="service.operation.name">
<EXPLAIN><TITLE>Service Operation Name</TITLE>
<P>The service operation name is a human-readable string that ascribes a moniker to a service
operation. It may be employed in user interfaces and documentation, or for other
purposes.</P></EXPLAIN>
<MODEL><STRING></STRING></MODEL>
</ELEMENTTYPE>

<H3>Service Operation Location</H3>
<INTRO><P>The service location is a network resource. That is to say, a URI.</P></INTRO>
<ELEMENTTYPE NAME="service.operation.location">
<EXPLAIN><TITLE>Service Operation Location</TITLE>
<SYNOPSIS>A URI of a service operation.</SYNOPSIS>
<P>A service operation location is a datatype-constrained string that locates a service operation
on the Internet by means of a URL.</P></EXPLAIN>
<MODEL><STRING DATATYPE="url"></STRING></MODEL>
</ELEMENTTYPE>

<H3>Service Operation Input Document</H3>
<INTRO><P>The input to a service operation is defined by its input document type. That is,
the service operation is invoked when the service operation location receives an input document
whose type corresponds to the document type specified by this element.</P>
<P>Rather than define the expected input and output document types in the market participant
document, this example provides pointers to externally-defined BIDs. This allows reuse of the
same BID as the input and/or output document type for multiple operations. In addition, it
encourages parallel design and implementation.</P></INTRO>
<ELEMENTTYPE NAME="service.operation.input">
```

```
<EXPLAIN><TITLE>Service Operation Input</TITLE>
<SYNOPSIS>Identifies the type of the service operation input document.</SYNOPSIS>
<P>Service location input is a datatype-constrained string that identifies a BID on the Internet
by means of a URI.</P>
</EXPLAIN>
<MODEL><STRING DATATYPE="url"></STRING></MODEL>
</ELEMENTTYPE>

<H3>Service Operation Output Document Type</H3>
<INTRO><P>The output of a service operation is defined by its output document type(s). That
is, the service operation is expected to emit a document whose type corresponds to the
document type specified by this element.</P></INTRO>
<ELEMENTTYPE NAME="service.operation.output">
<EXPLAIN><TITLE>Service Operation Output</TITLE>
<SYNOPSIS>Identifies the type of the service operation output document.</SYNOPSIS>
<P>Service location output is a datatype-constrained string that identifies a BID on the Internet
by means of a URI.</P>
</EXPLAIN>
<MODEL><STRING DATATYPE="url"></STRING></MODEL>
</ELEMENTTYPE>

<H3>Service Terms</H3>
<INTRO><P>This is a simple collection of string elements, describing the terms of an
agreement.</P></INTRO>
<ELEMENTTYPE NAME="service.terms">
<EXPLAIN><TITLE>Service Terms</TITLE>
<SYNOPSIS>Describes the terms of a given agreement.</SYNOPSIS>
</EXPLAIN>
<MODEL><STRING DATATYPE="string"></STRING></MODEL>
</ELEMENTTYPE>

</DTD>
</SCHEMA>
```

The service DTD schema may be extended with a service type element in a common business language repository as follows:

```
<!ELEMENT service.type EMPTY>
<!ATTLIST service.type
        service.type.name (
        catalog.operator
        | commercial.directory.operator
        | eft.services.provider
        | escrower
        | fulfillment.service
        | insurer
        | manufacturer
        | market.operator
```

```
| order.originator
| ordering.service
| personal.services.provider
| retailer
| retail.aggregator
| schema.resolution.service
| service.provider
| shipment.acceptor
| shipper
| van
| wholesale.aggregator
) #REQUIRED
%common.attrib;
>
```

The service type element above illustrates interpretation information carried by a business interface definition, in this example a content form allowing identification of any one of a list of valid service types. Other interpretation information includes data typing, such as for example the element <H3>Internet Address</H3> including the content form "url" and expressed in the data type "string." Yet other interpretation information includes mapping of codes to elements of a list, such as for example the element <H3>State</H3> including the code mapping for states in the file "COUNTRY.US.SUBENTITY."

The service description referred to by the market participant DTD defines the documents that the service accepts and generates upon competition of the service. A basic service description is specified below as a XML document transact.dtd.

Transact.dtd models a transaction description, such as an invoice, or a description of an exchange of value. This document type supports many uses, so the transaction description element has an attribute that allows user to distinguish among invoices, performance, offers to sell, requests for quotes and so on. The exchange may occur among more than two parties, but only two are called out, the offeror and the counter party, each of whom is represented by a

pointer to a document conforming to the market participant DTD outlined above. The counter party pointer is optional, to accommodate offers to sell. The exchange description is described in the module tranprim.mod listed below, and includes pricing and subtotals. Following the exchange description, charges applying to the transaction as a whole may be provided, and a total charge must be supplied. Thus, the transaction description schema document transact.dtd for this example is set forth below:

```
<!-- transact.dtd Version:  1.0 -->
<!-- Copyright 1998 Veo Systems, Inc. -->

...

<!ELEMENT transaction.description (meta?, issuer.pointer,
        counterparty.pointer?, exhange.descrption+, general.charges?,
        net.total?)>
<!ATTLIST transaction.description
        transaction.type (invoice | pro.forma | offer.to.sell | order
                | request.for.quote | request.for.bid
                | request.for.proposal | response.to.request.for.quote
                | response.to.request.for.bid
                | response.to.request.for.proposal) "invoice"
        %common.attrib;
        %altrep.attrib;
        %ttl.attrib;
>
```

Representative market participant, and service DTDs, created according to the definitions above, are as follows:

Market Participant DTD

```
<!ELEMENT business  (party.name+ , address.set)  >
<!ATTLIST business business.number  CDATA  #REQUIRED
        >
<!ELEMENT party.name (#PCDATA )>
<!ELEMENT city (#PCDATA )>
<!ELEMENT internet (#PCDATA )>
<!ELEMENT country (#PCDATA )>
<!ELEMENT state (#PCDATA )>
<!ELEMENT email (#PCDATA )>
<!ELEMENT address.physical  (street , city , state , postcode? , country)  >
<!ELEMENT telephone (#PCDATA )>
<!ELEMENT person  (party.name+ , address.set)  >
```

```
<!ATTLIST person SSN  CDATA  #IMPLIED
                >
<!ELEMENT fax (#PCDATA )>
<!ELEMENT street (#PCDATA )>
<!ELEMENT address.set  (address.physical , telephone* , fax* , email* , internet*) >
<!ELEMENT postcode (#PCDATA )>
<!ELEMENT market.participant  (business | person) >

Service DTD

<!ELEMENT service.location (#PCDATA )>
<!ELEMENT service.terms (#PCDATA )>
<!ELEMENT service.operation.name (#PCDATA )>
<!ELEMENT service.operation  (service.operation.name , service.operation.location ,
servicc.operation.input , service.operation.output) >
<!ELEMENT service  (service.name , service.location , service.operation+ , service.terms) >
<!ELEMENT service.operation.input (#PCDATA )>
<!ELEMENT service.operation.location (#PCDATA )>
<!ELEMENT service.name (#PCDATA )>
<!ELEMENT service.set (service+ )>
<!ELEMENT service.operation.output (#PCDATA )>
```

One instance of a document produced according to the transact.dtd follows:

```
<?xml version="1.0""?>
<!-- rorder.xml Version:  1.0 -->
<!-- Copyright 1998 Veo Systems, Inc. -->

<!DOCTYPE transaction.description SYSTEM "urn:x-
veosystems:dtd:cbl:transact:1.0:>
<transaction.description transaction.type="order">
<meta>
 <urn?urn:x-veosystems:doc:00023
 </urn>
        <thread.id party.assigned.by="reqorg">FRT876
        </thread.id>
</meta>
        <issuer.pointer>
                <xll.locator urllink="reqorg.xml">Customer
                Pointer
                </xll.locator>
        </issuer.pointer>
        <counterparty.pointer>
                <xll.locator urllink="compu.xml">Catalog entry owner
                pointer
                </xll.locator>
        </counterparty.pointer>
<exchange.description>
<line.item>
```

```
<product.instance>
<product.description.pointer>
        <xll.locator urllink="cthink.xml">Catalogue Entry    Pointer
        </xll.locator>
</product.description.pointer>
<product.specifics>
<info.description.set>
<info.description>
 <xml.descriptor>
        <doctype>
        <dtd system.id="urn:x-veosystems:dtd:cbl:gprod:1.0"/>
        </doctype>
        <xml.descriptor.details>
        <xll.xptr.frag>DESCENDANT(ALL,os)STRING("Windows
95")
        </xll.xptr.frag>
        <xll.xptr.frag>DECENDANT(ALL,p.speed)STRING("200")
        </xll.xptr.frag>
        <xll.xptr.frag>DESCENDANT(ALL,hard.disk.capacity)
                STRING("4")
        </xll.xptr.frag>
        <xll.xptr.frag>DESCENDANT(ALL,d.size)STRING("14.1")
        </xll.xptr.frag>
        </xml.descriptor.details>
 </xml.descriptor>
</info.description>
        </info.description.set>
        </product.specifics>
<quantity>1
</quantity>
        </product.instance>
<shipment.coordinates.set>
<shipment.coordinates>
<shipment.destination>
                <address.set>
        <address.named>SW-1
        </address.named>
        <address.physical>
                <building.sublocation>208C</building.sublocation>
                <location.in.street>123
                </location.in.street>
                <street>Frontage Rd.
                </street>
                <city>Beltway
                </city>
                <country.subentity.us
                country.subentity.us.name="MD"/>
                <postcode>20000
                </postcode>
        </address.physical>
```

```
                <telephone>
                        <telephone.number>617-666-2000
                        </telephone.number>
                        <telephone.extension>1201
                        </telephone.extension>
                </telephone>
                </address.set>
        </shipment.destination>

        <shipment.special>No deliveries after 4 PM</shipment.special>
        </shipment.coordinates>
        </shipment.coordinates.set>
                <payment.set>
                <credit.card
                issuer.name="VISA"
                instrument.number="3787-812345-67893"
                expiry.date="12/97"
                currency.code="USD"/>
                <amount.group>
                        <amount.monetary currency.code="USD">3975
                        </amount.monetary>
                </amount.group>
                </payment.set>
        </line.item>
        </exhange.description>
        </transaction.description>
```

Accordingly, the present invention provides a technique by which a market participant is able to identify itself, and identify the types of input documents and the types of output documents with which it is willing to transact business. The particular manner in which the content carried in such documents is processed by the other parties to the transaction, or by the local party, is not involved in establishing a business relationship nor carrying out transactions.

Fig. 3 provides a simplified view of a participant node in the network according to the present invention. The node illustrated in Fig. 3 includes a network interface 300 which is coupled to a communication network on port 301. The network interface is coupled to a document parser 301. The parser 301 supplies the logical structures from an incoming document to a translator module 302, which provides for translating the incoming document into a form

usable by the host transaction system, and vice versa translating the output of host processes into the format of a document which matches the output document form in the business interface definition for transmission to a destination. The parser 301 and translator 302 are responsive to the business interface definition stored in the participant module 303.

The output data structures from the translator 302 are supplied to a transaction process front end 304 along with events signaled by the parser 301. The front end 304 in one embodiment consists of a JAVA virtual machine or other similar interface adapted for communication amongst diverse nodes in a network. The transaction processing front end 304 responds to the events indicated by the parser 301 and the translator 302 to route the incoming data to appropriate functions in the enterprise systems and networks to which the participant is coupled. Thus, the transaction process front end 304 in the example of Fig. 3 is coupled to commercial functions 305, database functions 306, other enterprise functions such as accounting and billing 307, and to the specific event listeners and processors 308 which are designed to respond to the events indicated by the parser.

The parser 301 takes a purchase order like that in the example above, or other document, specified according to the business interface definition and creates a set of events that are recognized by the local transaction processing architecture, such as a set of JAVA events for a JAVA virtual machine.

The parser of the present invention is uncoupled from the programs that listen for events based on the received documents. Various pieces of mark-up in a received document or a complete document meeting certain specifications serve as instructions for listening functions to start processing. Thus listening programs carry out the business logic associated with the document information. For example, a program associated with an address element may be code that validates the postal code by checking the database. These listeners subscribe to

events by registering with a document router, which directs the relevant events to all subscribers who are interested in them.

For example, the purchase order specified above may be monitored by programs listening for events generated by the parser, which would connect the document or its contents to an order entry program. Receipt of product descriptions within the purchase order, might invoke a program to check inventory. Receipt of address information within the purchase order, would then invoke a program to check availability of services for delivery. Buyer information fields in the document, could invoke processes to check order history for credit worthiness or to offer a promotion or similar processing based on knowing the identity of the consumer.

Complex listeners can be created as configurations of primitive ones. For example, a purchase order listener may contain and invoke the list listeners set out in the previous paragraph, or the list members may be invoked on their own. Note that the applications that the listeners run are unlikely to be native XML processes or native JAVA processes. In these cases, the objects would be transformed into the format required by the receiving trans application. When the application finishes processing, its output is then transformed back to the XML format for communication to other nodes in the network.

It can be seen that the market participant document type description, and the transaction document type description outlined above include a schematic mapping for logic elements in the documents, and include mark-up language based on natural language. The natural language mark-up, and other natural language attributes of XML facilitate the use of XML type mark-up languages for the specification of business interface definitions, service descriptions, and the descriptions of input and output documents.

The participant module 303 in addition to storing the business interface definition includes a compiler which is used to compile objects or other data

structures to be used by the transaction process front end 304 which corresponds to the logical structures in the incoming documents, and to compile the translator 302. Thus, as the business interface definition is modified or updated by the participant as the transactions with which the participant is involved change, the translator 302 and parser 301 are automatically kept up to date.

In a preferred system, the set of JAVA events is created by a compiler which corresponds to the grove model of SGML, mainly the standard Element Structure Information Set augmented by the "property set" for each element. *International Standard ISO/IEC 10179:1996 (E), Information Technology -- Processing Languages -- Document Style Semantics and Specification Language (DSSSL)*. Turning the XML document into a set of events for the world to process contrasts with the normal model of parsing in which the parser output is maintained as an internal data structure. By translating the elements of the XML document into JAVA events or other programming structures that are suitable for use by the transaction processing front end of the respective nodes enables rich functionality at nodes utilizing the documents being traded.

Thus, the transaction process front end 304 is able to operate in a publish and subscribe architecture that enables the addition of new listener programs without the knowledge of or impact on other listening programs in the system. Each listener, 305, 306, 307, 308 in Fig. 3, maintains a queue in which the front end 304 directs events. This enables multiple listeners to handle events in parallel at their own pace.

Furthermore, according to the present invention the applications that the listeners run need not be native XML functions, or native functions which match the format of the incoming document. Rather, these listeners may be JAVA functions, if the transaction process front end 304 is a JAVA interface, or may be functions which run according to a unique transaction processing architecture. In these cases, the objects would be transformed into the format

required by the receiving application. When the application of the listener finishes, its output is then transformed back into the format of a document as specified by the business interface definition in the module 303. Thus, the translator 302 is coupled to the network interface 300 directly for supplying the composed documents as outputs.

The listeners coupled to the transaction processing front end may include listeners for input documents, listeners for specific elements of the input documents, and listeners for attributes stored in particular elements of the input document. This enables diverse and flexible implementations of transaction processes at the participant nodes for filtering and responding to incoming documents.

Fig. 4 illustrates a process of receiving and processing an incoming document for the system of Fig. 3. Thus, the process begins by receiving a document at the network interface (step 400). The parser identifies the document type (401) in response to the business interface definition. Using the business interface definition, which stores a DTD for the document in the XML format, the document is parsed (step 402). Next, the elements and attributes of the document are translated into the format of the host (step 403). In this example, the XML logic structures are translated into JAVA objects which carry the data of the XML element as well as methods associated with the data such as get and set functions. Next, the host objects are transferred to the host transaction processing front end (step 404). These objects are routed to processes in response to the events indicated by the parser and the translator. The processes which receive the elements of the document are executed and produce an output (step 405). The output is translated to the format of an output document as defined by the business interface definition (step 406). In this example, the translation proceeds from the form of a JAVA object to that of

an XML document. Finally, the output document is transmitted to its destination through the network interface (step 407).

Fig. 5 is a more detailed diagram of the event generator/event listener mechanism for the system of Fig. 3. In general the approach illustrated in Fig. 5 is a refinement of the JAVA JDK 1.1 event model. In this model, three kinds of objects are considered. A first kind of object is an event object which contains information about the occurrence of an event. There may be any number of kinds of event objects, corresponding to all the different kinds of events which can occur. A second kind of object is an Event generator, which monitors activity and generates event objects when something happens. Third, event listeners, listen for event objects generated by event generators. Event listeners generally listen to specific event generators, such as for mouse clicks on a particular window. Event listeners call an "ADD event listener" method on the event generator. This model can be adapted to the environment of Fig. 3 in which the objects are generated in response to parsing and walking a graph of objects, such as represented by an XML document.

The system illustrated in Fig. 5 includes a generic XML parser 500. Such parser can be implemented using a standard call back model. When a parsing event occurs, the parser calls a particular method in an application object, passing in the appropriate information in the parameters. Thus a single application 501 resides with the parser. The application packages the information provided by the parser in an XML event object and sends it to as many event listeners as have identified themselves, as indicated by the block 502. The set of events 502 is completely parser independent. The events 502 can be supplied to any number of listeners and any number of threads on any number of machines. The events are based on the element structure information set ESIS in one alternative. Thus, they consist of a list of the important aspects of a document structure, such as the starts and ends of elements, or of the

recognition of an attribute. XML (and SGML) parsers generally use the ESIS structure as a default set of information for a parser to return to its application.

A specialized ESIS listener 503 is coupled to the set of events 502. This listener 503 implements the ESIS listener API, and listens for all XML events from one or more generators. An element event generator 504 is a specialized ESIS listener which is also an XML event generator. Its listeners are objects only interested in events for particular types of elements. For example in an HTML environment, the listener may only be interested in ordered lists, that is only the part of the document between the <OL> and </OL>tags. For another example, a listener may listen for "party.name" elements, or for "service.name" elements according to the common business language, from the example documents above, process the events to ensure that the elements carry data that matches the schematic mapping for the element, and react according to the process needed at the receiving node.

This allows the system to have small objects that listen for particular parts of the document, such as one which only adds up prices. Since listeners can both add and remove themselves from a generator, there can be a listener which only listens to for example the <HEAD> part of an HTML document. Because of this and because of the highly recursive nature of XML documents, it is possible to write highly targeted code, and to write concurrent listeners. For example, an <OL> listener can set up an <LI> listener completely separate from the manner in which the <UL> (unordered list) listener sets up its <LI> listener. Alternatively, it can create a listener which generates a graphic user interface and another which searches a database using the same input. Thus, the document is treated as a program executed by the listeners, as opposed to the finished data structure which the application examines one piece at a time. If an application is written this way, it is not necessary to have the entire document in memory to execute an application.

The next listener coupled to the set of events 502 is an attribute filter 505. The attribute filter 505 like the element filter 504 is an attribute event generator according to the ESIS listener model. The listener for an attribute filter specifies the attributes it is interested in, and receives events for any element having that attribute specified. So for example, a font manager might receive events only for elements having a font attribute, such as the <P FONT= "Times Roman" /P>.

The element event generator 504 supplies such element objects to specialize the element listeners 504A.

The attribute event generator 505 supplies the attribute event objects to attribute listeners 505A. Similarly, the attribute objects are supplied to a "architecture" in the sense of an SGML/XML transformation from one document type to another using attributes. Thus the architecture of 505B allows a particular attribute with a particular name to be distinguished. Only elements with that attribute defined become part of the output document, and the name of the element in the output document is the value of the attribute in the input document. For example, if the architecture 505B is HTML, the string:

<PURCHASES HTML="OL"><ITEM HTML="LI"><NAME HTML="B">STUFF</NAME><PRICE HTML="B">123</PRICE></ITEM></PURCHASES>

translates into:

<OL><LI><B>STUFF</B><B>123</B></LI></OL>

which is correct HTML.

The next module which is coupled to the set of events 502 is a tree builder 506. The tree builder takes a stream of XML events and generates a tree representation of the underlying document. One preferred version of the tree builder 506 generates a document object model DOM object 507, according to the specification of the W3C (See, http://www.w3.org/TR/1998/

WD-DOM-19980720/ introduction.html). However listeners in event streams can be used to handle most requirements, a tree version is useful for supporting queries around a document, reordering of nodes, creation of new documents, and supporting a data structure in memory from which the same event stream can be generated multiple times, for example like parsing the document many times. A specialized builder 508 can be coupled to the tree builder 506 in order to build special subtrees for parts of the document as suits a particular implementation.

In addition to responses to incoming documents, other sources of XML events 502 can be provided. Thus, an event stream 510 is generated by walking over a tree of DOM objects and regenerating the original event stream created when the document was being parsed. This allows the system to present the appearance that the document is being parsed several times.

The idea of an object which walks a tree and generates a stream of events can be generalize beyond the tree of DOM objects, to any tree of objects which can be queried. Thus, a JAVA walker 512 may be an application which walks a tree of JAVA bean components 513. The walker walks over all the publicly accessible fields and methods. The walker keeps track of the objects it has already visited to ensure that it doesn't go into an endless cycle. JAVA events 514 are the type of events generated by the JAVA walker 512. This currently includes most of the kinds of information one can derive from an object. This is the JAVA equivalent of ESIS and allows the same programming approach applied to XML to be applied to JAVA objects generally, although particularly to JAVA beans.

The JAVA to XML event generator 515 constitutes a JAVA listener and a JAVA event generator. It receives the stream of events 514 from the JAVA walker 512 and translates selected ones to present a JAVA object as an XML document. In the one preferred embodiment, the event generator 515 exploits

the JAVA beans API. Each object seen becomes an element, with the element name the same as the class name. Within that element, each embedded method also becomes an element whose content is the value returned by invoking the method. If it is an object or an array of objects, then these are walked in turn.

Fig. 6 outlines a particular application built on the framework of Fig. 5. This application takes in an XML document 600 and applies it to a parser/generator 601. ESIS events 602 are generated and supplied to an attribute generator 603 and tree builder 604. The attribute generator corresponds to the generator 505 of Fig. 5. It supplies the events to the "architecture" 505B for translating the XML input to an HTML output for example. These events are processed in parallel as indicated by block 605 and processed by listeners. The output of the listeners are supplied to a document writer 506 and then translated back to an XML format for output. Thus for example this application illustrated in Fig. 6 takes an XML document and outputs an HTML document containing a form. The form is then sent to a browser, and the result is converted back to XML. For this exercise, the architecture concept provides the mapping from XML to HTML. The three architectures included in Fig. 6 include one for providing the structure of the HTML document, such as tables and lists, a second specifying text to be displayed, such as labels for input fields on the browser document, and the third describes the input fields themselves. The elements of the XML document required to maintain the XML documents structure become invisible fields in the HTML form. This is useful for use in reconstruction of the XML document from the information the client will put into the HTTP post message that is sent back to the server. Each architecture takes the input document and transforms it into an architecture based on a subset of HTML. Listeners listening for these events, output events for the HTML document, which then go to a document writer object. The document writer object listens to XML events and turns them

back into an XML document. The document writer object is a listener to all the element generators listening to the architectures in this example.

The organization of the processing module illustrated in Figs. 5 and 6 is representative of one embodiment of the parser and transaction process front end for the system of Fig. 3. As can be seen, a very flexible interface is provided by which diverse transaction processes can be executed in response to the incoming XML documents, or other structured document formats.

Fig. 7 illustrates a node similar to that of Fig. 3, except that it includes a business interface definition builder module 700. Thus, the system of Fig. 7 includes a network interface 701, a document parser 702, and a document translator 703. The translator 703 supplies its output to a transaction processing front end 704, which in turn is coupled to listening functions such as commercial functions 705, a database 706, enterprise functions 707, and other generic listeners and processors 708. As illustrated in Fig. 7, the business interface definition builder 700 includes a user interface, a common business library CBL repository, a process for reading complementary business interface definitions, and a compiler. The user interface is used to assist an enterprise in the building of a business interface definition relying on the common business library repository, and the ability to read complementary business interface definitions. Thus, the input document of a complementary business interface definition can be specified as the output document of a particular transaction, and the output document of the complementary business interface definition can be specified as the input to such transaction process. In a similar manner a transaction business interface definition can be composed using components selected from the CBL repository. The use of the CBL repository encourages the use of standardized document formats, such as the example schema (bid1) documents above, logical structures and interpretation information in the

building of business interface definitions which can be readily adopted by other people in the network.

The business interface definition builder module 700 also includes a compiler which is used for generating the translator 703, the objects to be produced by the translator according to the host transaction processing architecture, and to manage the parsing function 702.

Fig. 8 is a heuristic diagram showing logical structures stored in the repository in the business interface definition builder 700. Thus, the repository storage representative party business interface definitions 800, including for example a consumer BID 801, a catalog house BID 802, a warehouse BID 803, and an auction house BID 804. Thus, a new participant in an online market may select as a basic interface description one of the standardized BIDs which best matches its business. In addition, the repository will store a set of service business interface definitions 805. For example, an order entry BID 806, an order tracking BID 807, an order fulfillment BID 808, and a catalog service BID 809 could be stored. As a new participant in the market builds a business interface definition, it may select the business interface definitions of standardized services stored in the repository.

In addition to the party and service BIDs, input and output document BIDs are stored in the repository as indicated by the field 810. Thus, a purchase order BID 811, an invoice BID 812, a request for quote BID 813, a product availability report BID 814, and an order status BID 815 might be stored in the repository.

The repository, in addition to the business interface definitions which in a preferred system are specified as document type definitions according to XML, stores interpretation information in the form of semantic maps as indicated by the field 816. Thus, semantic maps which are used for specifying weights 817, currencies 818, sizes 819, product identifiers 820, and product

features 821 in this example might be stored in the repository. Further, the interpretation information provides for typing of data structures within the logical structures of documents.

In addition, logical structures used in the composing of business interface definitions could be stored in the repository as indicated by block 822. Thus, forms for providing address information 823, forms for providing pricing information 824, and forms for providing terms of contractual relationships could be provided 825. As the network expands, the CBL repository will also expand and standardize tending to make the addition of new participants, and the modification of business interface definitions easier.

Fig. 9 illustrates the process of building a business interface definition using the system of Fig. 7. The process begins by displaying a BID builder graphical interface to the user (step 900). The system accepts user input identifying a participant, service and document information generated by the graphical interface (step 901).

Next, any referenced logical structures, interpretation information, document definitions and/or service definitions are retrieved from the repository in response to user input via the graphical user interface (step 902). In the next step, any complementary business interface definitions or components of business interface definitions are accessed from other participants in the network selected via user input, by customized search engines, web browsers or otherwise (step 903). A document definition for the participant is created using the information gathered (step 904). The translators for the document to host and host to document mappers are created by the compiler (step 905). Host architecture data structures corresponding to the definition are created by the compiler (step 906), and the business interface definition which has been created is posted on the network, such as by posting on a website or otherwise, making it accessible to other nodes in the network (step 907).

Business interface definitions tell potential trading partners the online services the company offers and which documents to use to invoke those services. Thus, the services are defined in the business interface definition by the documents that they accept and produce. This is illustrated in the following fragment of an XML service definition.

```
<service>
<service.name>Order Service</service.name>
<service.location>www.veosystems.com/order</service.location>
<service.op>
<service.op.name>Submit Order</service.op.name>
<service.op.inputdoc>www.commerce.net/po.dtd</service.op.inputdoc>
<service.op.outputdoc>
        www.veosystems.com/invoice.dtd</service.op.outputdoc>
</service.op>
< service.op>
< service.op.name>Track Order</service.op.name>
<service.op.inputdoc> www.commerce.net
        /request.track.dtd<service.op.inputdoc>
<service.op.outputdoc>
        www.veosystems.com/response.track.dtd<service.op.outputdoc>
</service.op>
</service>
```

This XML fragment defines a service consisting of two transactions, one for taking orders and the other for tracking them. Each definition expresses a contract or promise to carry out a service if a valid request is submitted to the specified Web address. The Order service here requires an input document that conforms to a standard "po.dtd" Document Type Definition located in the repository, which may be local, or stored in an industry wide registry on the network. If a node can fulfill the order, it will return a document conforming to a customized "invoice.dtd" whose definition is local. In effect, the company is

promising to do business with anyone who can submit a Purchase Order that conforms to the XML specification it declares. No prior arrangement is necessary.

The DTD is the formal specification or grammar for documents of a given type; it describes the elements, their attributes, and the order in which they must appear. For example, purchase orders typically contain the names and addresses of the buyer and seller, a set of product descriptions, and associated terms and conditions such as price and delivery dates. In Electronic Data Interchange EDI for example, the X12 850 specification is a commonly used model for purchase orders.

The repository encourages the development of XML document models from reusable semantic components that are common to many business domains. Such documents can be understood from their common message elements, even though they may appear quite different. This is the role of the Common Business Library repository.

The Common Business Library repository consists of information models for generic business concepts including:

- business description primitives like companies, services, and products;
- business forms like catalogs, purchase orders, and invoices;
- standard measurements, date and time, location, classification codes.

This information is represented as an extensible, public set of XML building blocks that companies can customize and assemble to develop XML applications quickly. Atomic CBL elements implement industry messaging standards and conventions such as standard ISO codes for countries, currencies, addresses, and time. Low level CBL semantics have also come from analysis of proposed metadata frameworks for Internet resources, such as Dublin Core.

The next level of elements use these building blocks to implement the basic business forms such as those used in X12 EDI transactions as well as those used in emerging Internet standards such as OTP (Open Trading Protocol) and OBI (Open Buying on the Internet).

CBL's focus is on the functions and information that are common to all business domains (business description primitives like companies, services, and products; business forms like catalogs, purchase orders, and invoices; standard measurements, date and time, location, classification codes). CBL builds on standards or industry conventions for semantics where possible (e.g., the rules that specify "day/month/year" in Europe vs "month/day/year" in the U.S. are encoded in separate CBL modules).

The CBL is a language that is used for designing applications. It is designed to bridge the gap between the "document world" of XML and the "programming world" of JAVA or other transaction processing architectures. Schema embodies a philosophy of "programming with documents" in which a detailed formal specification of a document type is the master source from which a variety of related forms can be generated. These forms include XML DTDs for CBL, JAVA objects, programs for converting XML instances to and from the corresponding JAVA objects, and supporting documentation.

The CBL creates a single source from which almost all of the pieces of a system can be automatically generated by a compiler. The CBL works by extending SGML/XML, which is normally used to formally define the structures of particular document types, to include specification of the semantics associated with each information element and attribute. The limited set of (mostly) character types in SGML/XML can be extended to declare any kind of datatype.

Here is a fragment from the CBL definition for the "datetime" module:

```
<!-- datetime.mod Version: 1.0 -->
<!-- Copyright 1998 Veo Systems, Inc. -->
...
<! ELEMENT year (#PCDATA)>
<! ATTLIST year
        schema CDATA #FIXED "urn:x-veosystems:stds:iso:8601:3.8"
>

<! ELEMENT month (#PCDATA)>
<! ATTLIST month
        schema CDATA #FIXED "urn:x-veosystems:stds:iso:8601:3.12"
>

...
```

In this fragment, the ELEMENT "year" is defined as character data, and an associated "schema" attribute, also character data, defines the schema for "year" to be section 3.8 of the ISO 8601 standard.

This "datetime" CBL module is in fact defined as an instance of the Schema DTD. First, the module name is defined. Then the "datetime" element "YEAR" is bound to the semantics of ISO 8601:

```
<! DOCTYPE SCHEMA SYSTEM "schema.dtd">
<SCHEMA><H1>Date and Time Module</H1>

...
<ELEMNTTYPE NAME="year" DATATYPE="YEAR"><MODEL>
        <STRING
DATATYPE="YEAR"></STRING></MODEL>
<ATTDEF NAME=:schema:iso8601" DATATYPE="CDATA">
        <FIXED>3.8
Gregorian calendar</FIXED></ATTDEF></ELEMENTTYPE>

...
```

The example market participant and service modules above are also stored in the CBL repository.

In Fig. 10, an Airbill 1000 is being defined by customizing a generic purchase order DTD 1001, adding more specific information about shipping weight 1002. The generic purchase order 1001 was initially assembled from the

ground up out of CBL modules for address, date and time, currency, and vendor and product description. Using CBL thus significantly speeds the development and implementation of XML commerce applications. More importantly, CBL makes it easier for commercial applications to be interconnected.

In the CBL, XML is extended with a schema. The extensions add strong-typing to XML elements so that content can be readily validated. For example, an element called <CPU_clock_speed> can be defined as an integer with a set of valid values: {100, 133, 166, 200, 233, 266 Mhz.}. The schema also adds class-subclass hierarchies, so that information can be readily instantiated from class definitions. A laptop, for instance, can be described as a computer with additional tags for features such as display type and battery life. These and other extensions facilitate data entry, as well as automated translations between XML and traditional Object-Oriented and relational data models.

Thus the completed BID is run through the compiler which produces the DTDs for the actual instance of a participant and a service as outlined above, the JAVA beans which correspond to the logical structures in the DTD instances, and transformation code for transforming from XML to JAVA and from JAVA to XML. In alternative systems documentation is also generated for display on a user interface or for printing by a user to facilitate use of the objects.

For the example market participant and service DTDs set forth above, the JAVA beans generated by the compiler are set forth (with some redactions for conciseness) as follows:

```
import com.veo.vsp.doclet.meta.Document;
public class AddressPhysical extends Document {
    public static final String DOC_TYPE = "address.physical";
     String mStreet;
     String mCity;
    public final static int AK = 0;
    public final static int AL = 1;
    public final static int AR = 2;
    public final static int AZ = 3;
```

```
        public final static int CA = 4;
        ...

        public final static int WI = 48;
        public final static int WV = 49;
        public final static int WY = 50;
         int mState;
         String mPostcode;
        public final static int AD = 51;
        public final static int AE = 52;
        public final static int AF = 53;
        public final static int AG = 54;
        public final static int AI = 55;
        public final static int AM = 56;
        ...

         int mCountry;
        public AddressPhysical(){
                super(DOC_TYPE);
        mStreet = new String();
        mCity = new String();
                this.mState = -1;
        mPostcode = null;
                this.mCountry = -1;
        }
        public AddressPhysical(String doc_type){
                super(doc_type);
        mStreet = new String();
        mCity = new String();
                this.mState = -1;
        mPostcode = null;
                this.mCountry = -1;
        }
        static public AddressPhysical initAddressPhysical(String iStreet,String iCity,int
iState,String iPostcode,int iCountry){
                AddressPhysical obj = new AddressPhysical();
                obj.initializeAll(iStreet, iCity, iState, iPostcode, iCountry);
                return obj;
        }

        public void initializeAll(String iStreet,String iCity,int iState,String iPostcode,int
iCountry){
                mStreet = iStreet;
                mCity = iCity;
                mState = iState;
                mPostcode = iPostcode;
                mCountry = iCountry;
        }
        public String getStreet(){
         return mStreet;
```

```
}
public String getStreetToXML(){
        if (getStreet() == null) return null;
        char [] c = getStreet().toCharArray();
        StringBuffer sb = new StringBuffer();
        for (int x = 0; x < c.length; x++){
          switch(c[x]){
          case '>':
            sb.append("&gt;");
            break;
          case '<':
            sb.append("&lt;");
            break;
          case '&':
            sb.append("&amp;");
            break;
          case '"':
            sb.append("&quot;");
            break;
          case '\'':
            sb.append("&quot;");
            break;
          default:
            if (Character.isDefined(c[x]))
              sb.append(c[x]);
          }
        }
        return sb.toString();
}
public void setStreet(String inp){
        this.mStreet = inp;
}
public void streetFromXML(String n){
        setStreet(n);
}
public String getCity(){
 return mCity;
}
public String getCityToXML(){
        if (getCity() == null) return null;
        char [] c = getCity().toCharArray();
        StringBuffer sb = new StringBuffer();
        for (int x = 0; x < c.length; x++){
          switch(c[x]){
          case '>':
            sb.append("&gt;");
            break;
          case '<':
            sb.append("&lt;");
            break;
```

```
                    case '&':
                     sb.append("&amp;");
                     break;
                    case "":
                     sb.append("&quot;");
                     break;
                    case \":
                     sb.append("&quot;");
                     break;
                    default:
                     if (Character.isDefined(c[x]))
                      sb.append(c[x]);
                     }
                    }
                    return sb.toString();
            }
            public void setCity(String inp){
                    this.mCity = inp;
            }
            public void cityFromXML(String n){
                    setCity(n);
            }
            public int getState(){
             return mState;
            }
            public String getStateToXML(){
                    switch (mState){
                            case AK: return "AK";
                            case AL: return "AL";
                            case AR: return "AR";
                            case AZ: return "AZ";
                            ...

                    }
             return null;
            }
            public void setState(int inp){
                    this.mState = inp;
            }
            public void stateFromXML(String s){
                    if (s.equals("AK")) mState = AK;
                    else if (s.equals("AL"))mState = AL;
                    else if (s.equals("AR"))mState = AR;
                    else if (s.equals("AZ"))mState = AZ;
                    ...

            }
            public String getPostcode(){
             return mPostcode;
            }
```

```
public String getPostcodeToXML(){
        if (getPostcode() == null) return null;
        char [] c = getPostcode().toCharArray();
        StringBuffer sb = new StringBuffer();
        for (int x = 0; x < c.length; x++){
          switch(c[x]){
          case '>':
            sb.append("&gt;");
            break;
          case '<':
            sb.append("&lt;");
            break;
          case '&':
            sb.append("&amp;");
            break;
          case '"':
            sb.append("&quot;");
            break;
          case '\'':
            sb.append("&quot;");
            break;
          default:
            if (Character.isDefined(c[x]))
              sb.append(c[x]);
          }
        }
        return sb.toString();
}
public void setPostcode(String inp){
        this.mPostcode = inp;
}
public void postcodeFromXML(String n){
        setPostcode(n);
}
public int getCountry(){
 return mCountry;
}
public String getCountryToXML(){
        switch (mCountry){
                case AD: return "AD";
                case AE: return "AE";
                case AF: return "AF";
                ...

        }
 return null;
}
public void setCountry(int inp){
        this.mCountry = inp;
}
```

```
        public void countryFromXML(String s){
                if (s.equals("AD")) mCountry = AD;
                else if (s.equals("AE"))mCountry = AE;
                else if (s.equals("AF"))mCountry = AF;
                else if (s.equals("AG"))mCountry = AG;
                else if (s.equals("AI"))mCountry = AI;
                ..

        }
}

package com.veo.xdk.dev.schema.test.blib;

import com.veo.vsp.doclet.meta.Document;
public class AddressSet extends Document {
        public static final String DOC_TYPE = "address.set";
         AddressPhysical mAddressPhysical;
         String [] mTelephone;
         String [] mFax;
         String [] mEmail;
         String [] mInternet;
        public AddressSet(){
                super(DOC_TYPE);
                this.mAddressPhysical = new AddressPhysical();
        mTelephone = null;
        mFax = null;
        mEmail = null;
        mInternet = null;
        }
        public AddressSet(String doc_type){
                super(doc_type);
                this.mAddressPhysical = new AddressPhysical();
        mTelephone = null;
        mFax = null;
        mEmail = null;
        mInternet = null;
        }
        static public AddressSet initAddressSet(AddressPhysical iAddressPhysical,String []
iTelephone,String [] iFax,String [] iEmail,String [] iInternet){
                AddressSet obj = new AddressSet();
                obj.initializeAll(iAddressPhysical, iTelephone, iFax, iEmail, iInternet);
                return obj;
        }

        public void initializeAll(AddressPhysical iAddressPhysical,String [] iTelephone,String
[] iFax,String [] iEmail,String [] iInternet){
                mAddressPhysical = iAddressPhysical;
                mTelephone = iTelephone;
                mFax = iFax;
                mEmail = iEmail;
```

```
            mInternet = iInternet;
        }
        public AddressPhysical getAddressPhysical(){
         return mAddressPhysical;
        }
        public void setAddressPhysical(AddressPhysical inp){
            this.mAddressPhysical = inp;
        }
        public String [] getTelephone(){
         return mTelephone;
        }
        public String getTelephone(int index){
            if (this.mTelephone == null)
              return null;
            if (index >= this.mTelephone.length)
              return null;
            if (index < 0 && -index > this.mTelephone.length)
              return null;
            if (index >= 0) return this.mTelephone[index];
            return this.mTelephone[this.mTelephone.length + index];
        }
        public String [] getTelephoneToXML(){
            String [] valArr = getTelephone();
            if (valArr == null) return null;
            String [] nvArr = new String[valArr.length];
            for (int z = 0; z < nvArr.length; z++){
            char [] c = valArr[z].toCharArray();
            StringBuffer st = new StringBuffer();
            StringBuffer sb = new StringBuffer();
            for (int x = 0; x < c.length; x++){
              switch(c[x]){
              case '>':
                sb.append("&gt;");
                break;
              case '<':
                sb.append("&lt;");
                break;
              case '&':
                sb.append("&amp;");
                break;
              case '"':
                sb.append("&quot;");
                break;
              case '"':
                sb.append("&quot;");
                break;
              default:
                if (Character.isDefined(c[x]))
                  sb.append(c[x]);
              }
```

```
                    }
                    nvArr[z] = sb.toString();
                    }
                    return nvArr;
        }
        public void setTelephone(int index, String inp){
            if (this.mTelephone == null) {
                if (index < 0) {
                  this.mTelephone = new String[1];
                  this.mTelephone[0] = inp;
                } else {
                  this.mTelephone = new String[index + 1];
                  this.mTelephone[index] = inp;
                }
            } else if (index < 0) {
                        String [] newTelephone = new String[this.mTelephone.length + 1];
                        java.lang.System.arraycopy((Object)mTelephone, 0,
(Object)newTelephone, 0, this.mTelephone.length);
                        newTelephone[newTelephone.length - 1] = inp;
                        mTelephone = newTelephone;
                } else if (index >= this.mTelephone.length){
                        String [] newTelephone = new String[index + 1];
                        java.lang.System.arraycopy((Object)mTelephone, 0,
(Object)newTelephone, 0, this.mTelephone.length);
                        newTelephone[index] = inp;
                        mTelephone = newTelephone;
                } else {
                        this.mTelephone[index] = inp;
                }
        }
        public void setTelephone(String [] inp){
                this.mTelephone = inp;
        }
        public void telephoneFromXML(String n){
                setTelephone(-1, n);
        }
        public String [] getFax(){
          return mFax;
        }
        public String getFax(int index){
                if (this.mFax == null)
                  return null;
                if (index >= this.mFax.length)
                  return null;
                if (index < 0 && -index > this.mFax.length)
                  return null;
                if (index >= 0) return this.mFax[index];
                return this.mFax[this.mFax.length + index];
        }
        public String [] getFaxToXML(){
```

```
                String [] valArr = getFax();
                if (valArr == null) return null;
                String [] nvArr = new String[valArr.length];
                for (int z = 0; z < nvArr.length; z++){
                char [] c = valArr[z].toCharArray();
                StringBuffer st = new StringBuffer();
                StringBuffer sb = new StringBuffer();
                for (int x = 0; x < c.length; x++){
                 switch(c[x]){
                 case '>':
                  sb.append("&gt;");
                  break;
                 case '<':
                  sb.append("&lt;");
                  break;
                 case '&':
                  sb.append("&amp;");
                  break;
                 case '"':
                  sb.append("&quot;");
                  break;
                 case '"':
                  sb.append("&quot;");
                  break;
                 default:
                  if (Character.isDefined(c[x]))
                   sb.append(c[x]);
                 }
                }
                nvArr[z] = sb.toString();
                }
                return nvArr;
        }
        public void setFax(int index, String inp){
            if (this.mFax == null) {
                if (index < 0) {
                 this.mFax = new String[1];
                 this.mFax[0] = inp;
                } else {
                 this.mFax = new String[index + 1];
                 this.mFax[index] = inp;
                }
            } else if (index < 0) {
                        String [] newFax = new String[this.mFax.length + 1];
                        java.lang.System.arraycopy((Object)mFax, 0, (Object)newFax, 0,
this.mFax.length);
                        newFax[newFax.length - 1] = inp;
                        mFax = newFax;
                } else if (index >= this.mFax.length){
                        String [] newFax = new String[index + 1];
```

```
                    java.lang.System.arraycopy((Object)mFax, 0, (Object)newFax, 0,
this.mFax.length);
                    newFax[index] = inp;
                    mFax = newFax;
                } else {
                    this.mFax[index] = inp;
                }
            }
            public void setFax(String [] inp){
                this.mFax = inp;
            }
            public void faxFromXML(String n){
                setFax(-1, n);
            }
            public String [] getEmail(){
              return mEmail;
            }
            public String getEmail(int index){
                if (this.mEmail == null)
                  return null;
                if (index >= this.mEmail.length)
                  return null;
                if (index < 0 && -index > this.mEmail.length)
                  return null;
                if (index >= 0) return this.mEmail[index];
                return this.mEmail[this.mEmail.length + index];
            }
            public String [] getEmailToXML(){
                String [] valArr = getEmail();
                if (valArr == null) return null;
                String [] nvArr = new String[valArr.length];
                for (int z = 0; z < nvArr.length; z++){
                char [] c = valArr[z].toCharArray();
                StringBuffer st = new StringBuffer();
                StringBuffer sb = new StringBuffer();
                for (int x = 0; x < c.length; x++){
                  switch(c[x]){
                  case '>':
                    sb.append("&gt;");
                    break;
                  case '<':
                    sb.append("&lt;");
                    break;
                  case '&':
                    sb.append("&amp;");
                    break;
                  case '"':
                    sb.append("&quot;");
                    break;
                  case '\'':
```

```
                        sb.append("&quot;");
                        break;
                      default:
                        if (Character.isDefined(c[x]))
                          sb.append(c[x]);
                      }
                    }
                    nvArr[z] = sb.toString();
                    }
                    return nvArr;
                }
                public void setEmail(int index, String inp){
                    if (this.mEmail == null) {
                        if (index < 0) {
                          this.mEmail = new String[1];
                          this.mEmail[0] = inp;
                        } else {
                          this.mEmail = new String[index + 1];
                          this.mEmail[index] = inp;
                        }
                    } else if (index < 0) {
                                String [] newEmail = new String[this.mEmail.length + 1];
                                java.lang.System.arraycopy((Object)mEmail, 0, (Object)newEmail,
0, this.mEmail.length);
                                newEmail[newEmail.length - 1] = inp;
                                mEmail = newEmail;
                        } else if (index >= this.mEmail.length){
                                String [] newEmail = new String[index + 1];
                                java.lang.System.arraycopy((Object)mEmail, 0, (Object)newEmail,
0, this.mEmail.length);
                                newEmail[index] = inp;
                                mEmail = newEmail;
                        } else {
                                this.mEmail[index] = inp;
                        }
                }
                public void setEmail(String [] inp){
                        this.mEmail = inp;
                }
                public void emailFromXML(String n){
                        setEmail(-1, n);
                }
                public String [] getInternet(){
                  return mInternet;
                }
                public String getInternet(int index){
                        if (this.mInternet == null)
                          return null;
                        if (index >= this.mInternet.length)
                          return null;
```

```
                if (index < 0 && -index > this.mInternet.length)
                  return null;
                if (index >= 0) return this.mInternet[index];
                return this.mInternet[this.mInternet.length + index];
        }
        public String [] getInternetToXML(){
                String [] valArr = getInternet();
                if (valArr == null) return null;
                String [] nvArr = new String[valArr.length];
                for (int z = 0; z < nvArr.length; z++){
                char [] c = valArr[z].toCharArray();
                StringBuffer st = new StringBuffer();
                StringBuffer sb = new StringBuffer();
                for (int x = 0; x < c.length; x++){
                  switch(c[x]){
                  case '>':
                    sb.append("&gt;");
                    break;
                  case '<':
                    sb.append("&lt;");
                    break;
                  case '&':
                    sb.append("&amp;");
                    break;
                  case '"':
                    sb.append("&quot;");
                    break;
                  case '"':
                    sb.append("&quot;");
                    break;
                  default:
                    if (Character.isDefined(c[x]))
                      sb.append(c[x]);
                  }
                }
                nvArr[z] = sb.toString();
                }
                return nvArr;
        }
        public void setInternet(int index, String inp){
            if (this.mInternet == null) {
                  if (index < 0) {
                    this.mInternet = new String[1];
                    this.mInternet[0] = inp;
                  } else {
                    this.mInternet = new String[index + 1];
                    this.mInternet[index] = inp;
                  }
              } else if (index < 0) {
                        String [] newInternet = new String[this.mInternet.length + 1];
```

```
                    java.lang.System.arraycopy((Object)mInternet, 0,
(Object)newInternet, 0, this.mInternet.length);
                    newInternet[newInternet.length - 1] = inp;
                    mInternet = newInternet;
                } else if (index >= this.mInternet.length){
                    String [] newInternet = new String[index + 1];
                    java.lang.System.arraycopy((Object)mInternet, 0,
(Object)newInternet, 0, this.mInternet.length);
                    newInternet[index] = inp;
                    mInternet = newInternet;
                } else {
                    this.mInternet[index] = inp;
                }
        }
        public void setInternet(String [] inp){
            this.mInternet = inp;
        }
        public void internetFromXML(String n){
            setInternet(-1, n);
        }
}

package com.veo.xdk.dev.schema.test.blib;

import com.veo.vsp.doclet.meta.Document;
public class Business extends Party {
        public static final String DOC_TYPE = "business";
         String aBusinessNumber;
        public Business(){
            super(DOC_TYPE);
        aBusinessNumber = new String();
        }
        public Business(String doc_type){
            super(doc_type);
        aBusinessNumber = new String();
        }
        static public Business initBusiness(String iBusinessNumber,String []
iPartyName,AddressSet iAddressSet){
            Business obj = new Business();
            obj.initializeAll(iBusinessNumber, iPartyName, iAddressSet);
            return obj;
        }

        public void initializeAll(String iBusinessNumber,String [] iPartyName,AddressSet
iAddressSet){
            aBusinessNumber = iBusinessNumber;
            super.initializeAll(iPartyName, iAddressSet);
        }
        public String getBusinessNumber(){
          return aBusinessNumber;
```

```
        }
        public String getBusinessNumberToXML(){
                if (getBusinessNumber() == null) return null;
                char [] c = getBusinessNumber().toCharArray();
                StringBuffer sb = new StringBuffer();
                for (int x = 0; x < c.length; x++){
                  switch(c[x]){
                  case '>':
                    sb.append("&gt;");
                    break;
                  case '<':
                    sb.append("&lt;");
                    break;
                  case '&':
                    sb.append("&amp;");
                    break;
                  case '"':
                    sb.append("&quot;");
                    break;
                  case '\'':
                    sb.append("&quot;");
                    break;
                  default:
                    if (Character.isDefined(c[x]))
                      sb.append(c[x]);
                  }
                }
                return sb.toString();
        }
        public void setBusinessNumber(String inp){
                this.aBusinessNumber = inp;
        }
        public void businessNumberFromXML(String n){
                setBusinessNumber(n);
        }
}


import com.veo.vsp.doclet.meta.Document;
public class Party extends Document {
        public static final String DOC_TYPE = "party";
         String [] mPartyName;
         AddressSet mAddressSet;
        public Party(){
                super(DOC_TYPE);
        mPartyName = new String[0];
                this.mAddressSet = new AddressSet();
        }
        public Party(String doc_type){
                super(doc_type);
```

```
mPartyName = new String[0];
        this.mAddressSet = new AddressSet();
}
static public Party initParty(String [] iPartyName,AddressSet iAddressSet){
        Party obj = new Party();
        obj.initializeAll(iPartyName, iAddressSet);
        return obj;
}

public void initializeAll(String [] iPartyName,AddressSet iAddressSet){
        mPartyName = iPartyName;
        mAddressSet = iAddressSet;
}
public String [] getPartyName(){
 return mPartyName;
}
public String getPartyName(int index){
        if (this.mPartyName == null)
          return null;
        if (index >= this.mPartyName.length)
          return null;
        if (index < 0 && -index > this.mPartyName.length)
          return null;
        if (index >= 0) return this.mPartyName[index];
        return this.mPartyName[this.mPartyName.length + index];
}
public String [] getPartyNameToXML(){
        String [] valArr = getPartyName();
        if (valArr == null) return null;
        String [] nvArr = new String[valArr.length];
        for (int z = 0; z < nvArr.length; z++){
        char [] c = valArr[z].toCharArray();
        StringBuffer st = new StringBuffer();
        StringBuffer sb = new StringBuffer();
        for (int x = 0; x < c.length; x++){
         switch(c[x]){
         case '>':
          sb.append("&gt;");
          break;
         case '<':
          sb.append("&lt;");
          break;
         case '&':
          sb.append("&amp;");
          break;
         case '"':
          sb.append("&quot;");
          break;
         case '"':
          sb.append("&quot;");
```

```
                    break;
                   default:
                    if (Character.isDefined(c[x]))
                     sb.append(c[x]);
                   }
                  }
                  nvArr[z] = sb.toString();
                  }
                  return nvArr;
        }
        public void setPartyName(int index, String inp){
            if (this.mPartyName == null) {
                  if (index < 0) {
                   this.mPartyName = new String[1];
                   this.mPartyName[0] = inp;
                  } else {
                   this.mPartyName = new String[index + 1];
                   this.mPartyName[index] = inp;
                  }
              } else if (index < 0) {
                        String [] newPartyName = new String[this.mPartyName.length + 1];
                        java.lang.System.arraycopy((Object)mPartyName, 0,
(Object)newPartyName, 0, this.mPartyName.length);
                        newPartyName[newPartyName.length - 1] = inp;
                        mPartyName = newPartyName;
                  } else if (index >= this.mPartyName.length){
                        String [] newPartyName = new String[index + 1];
                        java.lang.System.arraycopy((Object)mPartyName, 0,
(Object)newPartyName, 0, this.mPartyName.length);
                        newPartyName[index] = inp;
                        mPartyName = newPartyName;
                  } else {
                        this.mPartyName[index]= inp;
                  }
        }
        public void setPartyName(String [] inp){
                  this.mPartyName = inp;
        }
        public void partyNameFromXML(String n){
                  setPartyName(-1, n);
        }
        public AddressSet getAddressSet(){
          return mAddressSet;
        }
        public void setAddressSet(AddressSet inp){
                  this.mAddressSet = inp;
        }
}

package com.veo.xdk.dev.schema.test.blib;
```

```
import com.veo.vsp.doclet.meta.Document;
public class Person extends Party {
        public static final String DOC_TYPE = "person";
         String aSSN;
        public Person(){
                super(DOC_TYPE);
        aSSN = null;
        }
        public Person(String doc_type){
                super(doc_type);
        aSSN = null;
        }
        static public Person initPerson(String iSSN,String [] iPartyName,AddressSet
iAddressSet){
                Person obj = new Person();
                obj.initializeAll(iSSN, iPartyName, iAddressSet);
                return obj;
        }

        public void initializeAll(String iSSN,String [] iPartyName,AddressSet iAddressSet){
                aSSN = iSSN;
                super.initializeAll(iPartyName, iAddressSet);
        }
        public String getSSN(){
         return aSSN;
        }
        public String getSSNToXML(){
                if (getSSN() == null) return null;
                char [] c = getSSN().toCharArray();
                StringBuffer sb = new StringBuffer();
                for (int x = 0; x < c.length; x++){
                 switch(c[x]){
                 case '>':
                  sb.append("&gt;");
                  break;
                 case '<':
                  sb.append("&lt;");
                  break;
                 case '&':
                  sb.append("&amp;");
                  break;
                 case '"':
                  sb.append("&quot;");
                  break;
                 case '\'':
                  sb.append("&quot;");
                  break;
                 default:
                  if (Character.isDefined(c[x]))
                   sb.append(c[x]);
```

```
                }
            }
            return sb.toString();
        }
        public void setSSN(String inp){
            this.aSSN = inp;
        }
        public void sSNFromXML(String n){
            setSSN(n);
        }
}

package com.veo.xdk.dev.schema.test.blib;

import com.veo.vsp.doclet.meta.Document;
public class PrototypeService extends Document {
        public static final String DOC_TYPE = "prototype.service";
         String mServiceName;
         String [] mServiceTerms;
         String [] mServiceLocation;
         ServiceOperation [] mServiceOperation;
        public PrototypeService(){
            super(DOC_TYPE);
        mServiceName = new String();
        mServiceTerms = new String[0];
        mServiceLocation = new String[0];
            this.mServiceOperation = new ServiceOperation[0];
        }
        public PrototypeService(String doc_type){
            super(doc_type);
        mServiceName = new String();
        mServiceTerms = new String[0];
        mServiceLocation = new String[0];
            this.mServiceOperation = new ServiceOperation[0];
        }
        static public PrototypeService initPrototypeService(String iServiceName,String []
iServiceTerms,String [] iServiceLocation,ServiceOperation [] iServiceOperation){
            PrototypeService obj = new PrototypeService();
            obj.initializeAll(iServiceName, iServiceTerms, iServiceLocation,
iServiceOperation);
            return obj;
        }

        public void initializeAll(String iServiceName,String [] iServiceTerms,String []
iServiceLocation,ServiceOperation [] iServiceOperation){
            mServiceName = iServiceName;
            mServiceTerms = iServiceTerms;
            mServiceLocation = iServiceLocation;
            mServiceOperation = iServiceOperation;
        }
```

```
public String getServiceName(){
 return mServiceName;
}
public String getServiceNameToXML(){
        if (getServiceName() == null) return null;
        char [] c = getServiceName().toCharArray();
        StringBuffer sb = new StringBuffer();
        for (int x = 0; x < c.length; x++){
          switch(c[x]){
          case '>':
            sb.append("&gt;");
            break;
          case '<':
            sb.append("&lt;");
            break;
          case '&':
            sb.append("&amp;");
            break;
          case '"':
            sb.append("&quot;");
            break;
          case '\'':
            sb.append("&quot;");
            break;
          default:
            if (Character.isDefined(c[x]))
              sb.append(c[x]);
          }
        }
        return sb.toString();
}
public void setServiceName(String inp){
        this.mServiceName = inp;
}
public void serviceNameFromXML(String n){
        setServiceName(n);
}
public String [] getServiceTerms(){
 return mServiceTerms;
}
public String getServiceTerms(int index){
        if (this.mServiceTerms == null)
          return null;
        if (index >= this.mServiceTerms.length)
          return null;
        if (index < 0 && -index > this.mServiceTerms.length)
          return null;
        if (index >= 0) return this.mServiceTerms[index];
        return this.mServiceTerms[this.mServiceTerms.length + index];
}
```

```
public String [] getServiceTermsToXML(){
        String [] valArr = getServiceTerms();
        if (valArr == null) return null;
        String [] nvArr = new String[valArr.length];
        for (int z = 0; z < nvArr.length; z++){
        char [] c = valArr[z].toCharArray();
        StringBuffer st = new StringBuffer();
        StringBuffer sb = new StringBuffer();
        for (int x = 0; x < c.length; x++){
          switch(c[x]){
          case '>':
            sb.append("&gt;");
            break;
          case '<':
            sb.append("&lt;");
            break;
          case '&':
            sb.append("&amp;");
            break;
          case '"':
            sb.append("&quot;");
            break;
          case '"':
            sb.append("&quot;");
            break;
          default:
            if (Character.isDefined(c[x]))
              sb.append(c[x]);
          }
        }
        nvArr[z] = sb.toString();
        }
        return nvArr;
}
public void setServiceTerms(int index, String inp){
    if (this.mServiceTerms == null) {
        if (index < 0) {
          this.mServiceTerms = new String[1];
          this.mServiceTerms[0] = inp;
        } else {
          this.mServiceTerms = new String[index + 1];
          this.mServiceTerms[index] = inp;
        }
    } else if (index < 0) {
                String [] newServiceTerms = new String[this.mServiceTerms.length
+ 1];
                java.lang.System.arraycopy((Object)mServiceTerms, 0,
(Object)newServiceTerms, 0, this.mServiceTerms.length);
                newServiceTerms[newServiceTerms.length - 1] = inp;
                mServiceTerms = newServiceTerms;
```

```
                } else if (index >= this.mServiceTerms.length){
                        String [] newServiceTerms = new String[index + 1];
                        java.lang.System.arraycopy((Object)mServiceTerms, 0,
(Object)newServiceTerms, 0, this.mServiceTerms.length);
                        newServiceTerms[index] = inp;
                        mServiceTerms = newServiceTerms;
                } else {
                        this.mServiceTerms[index] = inp;
                }
        }
        public void setServiceTerms(String [] inp){
                this.mServiceTerms = inp;
        }
        public void serviceTermsFromXML(String n){
                setServiceTerms(-1, n);
        }
        public String [] getServiceLocation(){
         return mServiceLocation;
        }
        public String getServiceLocation(int index){
                if (this.mServiceLocation == null)
                  return null;
                if (index >= this.mServiceLocation.length)
                  return null;
                if (index < 0 && -index > this.mServiceLocation.length)
                  return null;
                if (index >= 0) return this.mServiceLocation[index];
                return this.mServiceLocation[this.mServiceLocation.length + index];
        }
        public String [] getServiceLocationToXML(){
                String [] valArr = getServiceLocation();
                if (valArr == null) return null;
                String [] nvArr = new String[valArr.length];
                for (int z = 0; z < nvArr.length; z++){
                char [] c = valArr[z].toCharArray();
                StringBuffer st = new StringBuffer();
                StringBuffer sb = new StringBuffer();
                for (int x = 0; x < c.length; x++){
                  switch(c[x]){
                  case '>':
                    sb.append("&gt;");
                    break;
                  case '<':
                    sb.append("&lt;");
                    break;
                  case '&':
                    sb.append("&amp;");
                    break;
                  case '"':
                    sb.append("&quot;");
```

```
                        break;
                       case "":
                        sb.append("&amp;quot;");
                        break;
                       default:
                        if (Character.isDefined(c[x]))
                         sb.append(c[x]);
                       }
                      }
                      nvArr[z] = sb.toString();
                      }
                      return nvArr;
                }
                public void setServiceLocation(int index, String inp){
                    if (this.mServiceLocation == null) {
                        if (index < 0) {
                         this.mServiceLocation = new String[1];
                         this.mServiceLocation[0] = inp;
                        } else {
                         this.mServiceLocation = new String[index + 1];
                         this.mServiceLocation[index] = inp;
                        }
                    } else if (index < 0) {
                                String [] newServiceLocation = new
String[this.mServiceLocation.length + 1];
                                java.lang.System.arraycopy((Object)mServiceLocation, 0,
(Object)newServiceLocation, 0, this.mServiceLocation.length);
                                newServiceLocation[newServiceLocation.length - 1] = inp;
                                mServiceLocation = newServiceLocation;
                        } else if (index >= this.mServiceLocation.length){
                                String [] newServiceLocation = new String[index + 1];
                                java.lang.System.arraycopy((Object)mServiceLocation, 0,
(Object)newServiceLocation, 0, this.mServiceLocation.length);
                                newServiceLocation[index] = inp;
                                mServiceLocation = newServiceLocation;
                        } else {
                                this.mServiceLocation[index] = inp;
                        }
                }
                public void setServiceLocation(String [] inp){
                        this.mServiceLocation = inp;
                }
                public void serviceLocationFromXML(String n){
                        setServiceLocation(-1, n);
                }
                public ServiceOperation [] getServiceOperation(){
                 return mServiceOperation;
                }
                public ServiceOperation getServiceOperation(int index){
                        if (this.mServiceOperation == null)
```

```
                return null;
            if (index >= this.mServiceOperation.length)
                return null;
            if (index < 0 && -index > this.mServiceOperation.length)
                return null;
            if (index >= 0) return this.mServiceOperation[index];
            return this.mServiceOperation[this.mServiceOperation.length + index];
        }
        public void setServiceOperation(int index, ServiceOperation inp){
            if (this.mServiceOperation == null) {
                if (index < 0) {
                this.mServiceOperation = new ServiceOperation[1];
                this.mServiceOperation[0] = inp;
                } else {
                this.mServiceOperation = new ServiceOperation[index + 1];
                this.mServiceOperation[index] = inp;
                }
            } else if (index < 0) {
                        ServiceOperation [] newServiceOperation = new
ServiceOperation[this.mServiceOperation.length + 1];
                        java.lang.System.arraycopy((Object)mServiceOperation, 0,
(Object)newServiceOperation, 0, this.mServiceOperation.length);
                        newServiceOperation[newServiceOperation.length - 1] = inp;
                        mServiceOperation = newServiceOperation;
            } else if (index >= this.mServiceOperation.length){
                        ServiceOperation [] newServiceOperation = new
ServiceOperation[index + 1];
                        java.lang.System.arraycopy((Object)mServiceOperation, 0,
(Object)newServiceOperation, 0, this.mServiceOperation.length);
                        newServiceOperation[index] = inp;
                        mServiceOperation = newServiceOperation;
            } else {
                        this.mServiceOperation[index] = inp;
            }
        }
        public void setServiceOperation(ServiceOperation [] inp){
            this.mServiceOperation = inp;
        }
}

package com.veo.xdk.dev.schema.test.blib;

import com.veo.vsp.doclet.meta.Document;
public class Service extends Document {
        public static final String DOC_TYPE = "service";
        String mServiceName;
        String mServiceLocation;
        ServiceOperation [] mServiceOperation;
        String mServiceTerms;
        public Service(){
```

```
                super(DOC_TYPE);
        mServiceName = new String();
        mServiceLocation = new String();
                this.mServiceOperation = new ServiceOperation[0];
        mServiceTerms = new String();
        }
        public Service(String doc_type){
                super(doc_type);
        mServiceName = new String();
        mServiceLocation = new String();
                this.mServiceOperation = new ServiceOperation[0];
        mServiceTerms = new String();
        }
        static public Service initService(String iServiceName,String
iServiceLocation,ServiceOperation [] iServiceOperation,String iServiceTerms){
                Service obj = new Service();
                obj.initializeAll(iServiceName, iServiceLocation, iServiceOperation,
iServiceTerms);
                return obj;
        }

        public void initializeAll(String iServiceName,String
iServiceLocation,ServiceOperation [] iServiceOperation,String iServiceTerms){
                mServiceName = iServiceName;
                mServiceLocation = iServiceLocation;
                mServiceOperation = iServiceOperation;
                mServiceTerms = iServiceTerms;
        }
        public String getServiceName(){
          return mServiceName;
        }
        public String getServiceNameToXML(){
                if (getServiceName() == null) return null;
                char [] c = getServiceName().toCharArray();
                StringBuffer sb = new StringBuffer();
                for (int x = 0; x < c.length; x++){
                  switch(c[x]){
                  case '>':
                    sb.append("&gt;");
                    break;
                  case '<':
                    sb.append("&lt;");
                    break;
                  case '&':
                    sb.append("&amp;");
                    break;
                  case '"':
                    sb.append("&quot;");
                    break;
                  case '\':
```

```
                    sb.append("&quot;");
                    break;
                  default:
                    if (Character.isDefined(c[x]))
                      sb.append(c[x]);
                  }
                 }
                 return sb.toString();
        }
        public void setServiceName(String inp){
                 this.mServiceName = inp;
        }
        public void serviceNameFromXML(String n){
                 setServiceName(n);
        }
        public String getServiceLocation(){
         return mServiceLocation;
        }
        public String getServiceLocationToXML(){
                 if (getServiceLocation() == null) return null;
                 char [] c = getServiceLocation().toCharArray();
                 StringBuffer sb = new StringBuffer();
                 for (int x = 0; x < c.length; x++){
                  switch(c[x]){
                  case '>':
                    sb.append("&gt;");
                    break;
                  case '<':
                    sb.append("&lt;");
                    break;
                  case '&':
                    sb.append("&amp;");
                    break;
                  case '"':
                    sb.append("&quot;");
                    break;
                  case '\'':
                    sb.append("&quot;");
                    break;
                  default:
                    if (Character.isDefined(c[x]))
                      sb.append(c[x]);
                  }
                 }
                 return sb.toString();
        }
        public void setServiceLocation(String inp){
                 this.mServiceLocation = inp;
        }
        public void serviceLocationFromXML(String n){
```

```
                setServiceLocation(n);
        }
        public ServiceOperation [] getServiceOperation(){
          return mServiceOperation;
        }
        public ServiceOperation getServiceOperation(int index){
                if (this.mServiceOperation == null)
                  return null;
                if (index >= this.mServiceOperation.length)
                  return null;
                if (index < 0 && -index > this.mServiceOperation.length)
                  return null;
                if (index >= 0) return this.mServiceOperation[index];
                return this.mServiceOperation[this.mServiceOperation.length + index];
        }
        public void setServiceOperation(int index, ServiceOperation inp){
            if (this.mServiceOperation == null) {
                if (index < 0) {
                  this.mServiceOperation = new ServiceOperation[1];
                  this.mServiceOperation[0] = inp;
                } else {
                  this.mServiceOperation = new ServiceOperation[index + 1];
                  this.mServiceOperation[index] = inp;
                }
            } else if (index < 0) {
                        ServiceOperation [] newServiceOperation = new
ServiceOperation[this.mServiceOperation.length + 1];
                        java.lang.System.arraycopy((Object)mServiceOperation, 0,
(Object)newServiceOperation, 0, this.mServiceOperation.length);
                        newServiceOperation[newServiceOperation.length - 1] = inp;
                        mServiceOperation = newServiceOperation;
                } else if (index >= this.mServiceOperation.length){
                        ServiceOperation [] newServiceOperation = new
ServiceOperation[index + 1];
                        java.lang.System.arraycopy((Object)mServiceOperation, 0,
(Object)newServiceOperation, 0, this.mServiceOperation.length);
                        newServiceOperation[index] = inp;
                        mServiceOperation = newServiceOperation;
                } else {
                        this.mServiceOperation[index] = inp;
                }
        }
        public void setServiceOperation(ServiceOperation [] inp){
                this.mServiceOperation = inp;
        }
        public String getServiceTerms(){
          return mServiceTerms;
        }
        public String getServiceTermsToXML(){
                if (getServiceTerms() == null) return null;
```

```
                    char [] c = getServiceTerms().toCharArray();
                    StringBuffer sb = new StringBuffer();
                    for (int x = 0; x < c.length; x++){
                     switch(c[x]){
                     case '>':
                      sb.append("&gt;");
                      break;
                     case '<':
                      sb.append("&lt;");
                      break;
                     case '&':
                      sb.append("&amp;");
                      break;
                     case "":
                      sb.append("&quot;");
                      break;
                     case '\":
                      sb.append("&quot;");
                      break;
                     default:
                      if (Character.isDefined(c[x]))
                       sb.append(c[x]);
                     }
                    }
                    return sb.toString();
            }
            public void setServiceTerms(String inp){
                    this.mServiceTerms = inp;
            }
            public void serviceTermsFromXML(String n){
                    setServiceTerms(n);
            }
    }

    package com.veo.xdk.dev.schema.test.blib;

    import com.veo.vsp.doclet.meta.Document;
    public class ServiceOperation extends Document {
            public static final String DOC_TYPE = "service.operation";
             String mServiceOperationName;
             String mServiceOperationLocation;
             String mServiceOperationInput;
             String mServiceOperationOutput;
             public ServiceOperation(){
                    super(DOC_TYPE);
             mServiceOperationName = new String();
             mServiceOperationLocation = new String();
             mServiceOperationInput = new String();
             mServiceOperationOutput = new String();
             }
```

```
        public ServiceOperation(String doc_type){
                super(doc_type);
        mServiceOperationName = new String();
        mServiceOperationLocation = new String();
        mServiceOperationInput = new String();
        mServiceOperationOutput = new String();
        }
        static public ServiceOperation initServiceOperation(String
iServiceOperationName,String iServiceOperationLocation,String iServiceOperationInput,String
iServiceOperationOutput){
                ServiceOperation obj = new ServiceOperation();
                obj.initializeAll(iServiceOperationName, iServiceOperationLocation,
iServiceOperationInput, iServiceOperationOutput);
                return obj;
        }

        public void initializeAll(String iServiceOperationName,String
iServiceOperationLocation,String iServiceOperationInput,String iServiceOperationOutput){
                mServiceOperationName = iServiceOperationName;
                mServiceOperationLocation = iServiceOperationLocation;
                mServiceOperationInput = iServiceOperationInput;
                mServiceOperationOutput = iServiceOperationOutput;
        }
        public String getServiceOperationName(){
          return mServiceOperationName;
        }
        public String getServiceOperationNameToXML(){
                if (getServiceOperationName() == null) return null;
                char [] c = getServiceOperationName().toCharArray();
                StringBuffer sb = new StringBuffer();
                for (int x = 0; x < c.length; x++){
                 switch(c[x]){
                 case '>':
                  sb.append("&gt;");
                  break;
                 case '<':
                  sb.append("&lt;");
                  break;
                 case '&':
                  sb.append("&amp;");
                  break;
                 case '"':
                  sb.append("&quot;");
                  break;
                 case '\'':
                  sb.append("&quot;");
                  break;
                 default:
                  if (Character.isDefined(c[x]))
                   sb.append(c[x]);
```

```
                }
              }
              return sb.toString();
        }
        public void setServiceOperationName(String inp){
              this.mServiceOperationName = inp;
        }
        public void serviceOperationNameFromXML(String n){
              setServiceOperationName(n);
        }
        public String getServiceOperationLocation(){
         return mServiceOperationLocation;
        }
        public String getServiceOperationLocationToXML(){
              if (getServiceOperationLocation() == null) return null;
              char [] c = getServiceOperationLocation().toCharArray();
              StringBuffer sb = new StringBuffer();
              for (int x = 0; x < c.length; x++){
               switch(c[x]){
               case '>':
                sb.append("&gt;");
                break;
               case '<':
                sb.append("&lt;");
                break;
               case '&':
                sb.append("&amp;");
                break;
               case '"':
                sb.append("&quot;");
                break;
               case '\':
                sb.append("&quot;");
                break;
               default:
                if (Character.isDefined(c[x]))
                  sb.append(c[x]);
               }
              }
              return sb.toString();
        }
        public void setServiceOperationLocation(String inp){
              this.mServiceOperationLocation = inp;
        }
        public void serviceOperationLocationFromXML(String n){
              setServiceOperationLocation(n);
        }
        public String getServiceOperationInput(){
         return mServiceOperationInput;
        }
```

```
public String getServiceOperationInputToXML(){
        if (getServiceOperationInput() == null) return null;
        char [] c = getServiceOperationInput().toCharArray();
        StringBuffer sb = new StringBuffer();
        for (int x = 0; x < c.length; x++){
          switch(c[x]){
          case '>':
            sb.append("&gt;");
            break;
          case '<':
            sb.append("&lt;");
            break;
          case '&':
            sb.append("&amp;");
            break;
          case '"':
            sb.append("&quot;");
            break;
          case '\'':
            sb.append("&quot;");
            break;
          default:
            if (Character.isDefined(c[x]))
              sb.append(c[x]);
          }
        }
        return sb.toString();
}
public void setServiceOperationInput(String inp){
        this.mServiceOperationInput = inp;
}
public void serviceOperationInputFromXML(String n){
        setServiceOperationInput(n);
}
public String getServiceOperationOutput(){
  return mServiceOperationOutput;
}
public String getServiceOperationOutputToXML(){
        if (getServiceOperationOutput() == null) return null;
        char [] c = getServiceOperationOutput().toCharArray();
        StringBuffer sb = new StringBuffer();
        for (int x = 0; x < c.length; x++){
          switch(c[x]){
          case '>':
            sb.append("&gt;");
            break;
          case '<':
            sb.append("&lt;");
            break;
          case '&':
```

```
                        sb.append("&amp;");
                        break;
                      case "":
                        sb.append("&quot;");
                        break;
                      case "\":
                        sb.append("&quot;");
                        break;
                      default:
                        if (Character.isDefined(c[x]))
                          sb.append(c[x]);
                      }
                    }
                    return sb.toString();
        }
        public void setServiceOperationOutput(String inp){
                this.mServiceOperationOutput = inp;
        }
        public void serviceOperationOutputFromXML(String n){
                setServiceOperationOutput(n);
        }
}

package com.veo.xdk.dev.schema.test.blib;

import com.veo.vsp.doclet.meta.Document;
public class ServiceSet extends Document {
        public static final String DOC_TYPE = "service.set";
         Service [] mService;
        public ServiceSet(){
                super(DOC_TYPE);
                this.mService = new Service[0];
        }
        public ServiceSet(String doc_type){
                super(doc_type);
                this.mService = new Service[0];
        }
        static public ServiceSet initServiceSet(Service [] iService){
                ServiceSet obj = new ServiceSet();
                obj.initializeAll(iService);
                return obj;
        }

        public void initializeAll(Service [] iService){
                mService = iService;
        }
        public Service [] getService(){
          return mService;
        }
        public Service getService(int index){
```

```
                    if (this.mService == null)
                      return null;
                    if (index >= this.mService.length)
                      return null;
                    if (index < 0 && -index > this.mService.length)
                      return null;
                    if (index >= 0) return this.mService[index];
                    return this.mService[this.mService.length + index];
                }
                public void setService(int index, Service inp){
                    if (this.mService == null) {
                        if (index < 0) {
                          this.mService = new Service[1];
                          this.mService[0] = inp;
                        } else {
                          this.mService = new Service[index + 1];
                          this.mService[index] = inp;
                        }
                    } else if (index < 0) {
                                Service [] newService = new Service[this.mService.length + 1];
                                java.lang.System.arraycopy((Object)mService, 0,
(Object)newService, 0, this.mService.length);
                                newService[newService.length - 1] = inp;
                                mService = newService;
                    } else if (index >= this.mService.length){
                                Service [] newService = new Service[index + 1];
                                java.lang.System.arraycopy((Object)mService, 0,
(Object)newService, 0, this.mService.length);
                                newService[index] = inp;
                                mService = newService;
                    } else {
                                this.mService[index] = inp;
                    }
                }
                public void setService(Service [] inp){
                    this.mService = inp;
                }
}
```

In addition to the JAVA beans set forth above, transformation code is produced for translating from JAVA to XML and XML to JAVA as set forth below:

Java to XML

```
<!DOCTYPE tree SYSTEM "tree.dtd">
<tree source = "null" pass-through = "false">
<before>
 <vardef name = "attribute.def">
```

```
<element source = "ATTRIBUTE" class = "NAME" type = "5" position = "-2">
 <parse>
  <data class = "java.lang.String" position = "-2"/>
 </parse>
</element>
</vardef>
<vardef name = "pcdata.def">
 <element source = "PCDATA" class = "NAME" type = "4" position = "-2">
  <parse>
   <data class = "999" type = "6" position = "-2"/>
  </parse>
 </element>
</vardef>
<vardef name = "content.def">
 <element source = "PCDATA">
  <parse>
   <data class = "999" type = "6" position = "-2"/>
  </parse>
 </element>
</vardef>
<vardef name = "ServiceSet.var">
 <element source = "com.veo.xdk.dev.schema.test.blib.ServiceSet" class = "service.set" type =
"4" position = "-2">
  <parse>
   <callvar name = "Service.var"/>
  </parse>
 </element>
</vardef>
<vardef name = "PrototypeService.var">
 <element source = "com.veo.xdk.dev.schema.test.blib.PrototypeService" class =
"prototype.service" type = "4" position = "-2">
  <parse>
   <callvar name = "pcdata.def" parms = "setSource ServiceNameToXML setGenerator
service.name"/>
   <callvar name = "pcdata.def" parms = "setSource ServiceTermsToXML setGenerator
service.terms"/>
   <callvar name = "pcdata.def" parms = "setSource ServiceLocationToXML setGenerator
service.location"/>
   <callvar name = "ServiceOperation.var"/>
  </parse>
 </element>
</vardef>
<vardef name = "Service.var">
 <element source = "com.veo.xdk.dev.schema.test.blib.Service" class = "service" type = "8"
position = "0">
  <parse>
   <callvar name = "pcdata.def" parms = "setSource ServiceNameToXML setGenerator
service.name"/>
   <callvar name = "pcdata.def" parms = "setSource ServiceLocationToXML setGenerator
service.location"/>
```

```
    <callvar name = "ServiceOperation.var"/>
    <callvar name = "pcdata.def" parms = "setSource ServiceTermsToXML setGenerator
service.terms"/>
   </parse>
  </element>
  </vardef>
  <vardef name = "ServiceOperation.var">
  <element source = "com.veo.xdk.dev.schema.test.blib.ServiceOperation" class =
"service.operation" type = "4" position = "-2">
    <parse>
     <callvar name = "pcdata.def" parms = "setSource ServiceOperationNameToXML
setGenerator service.operation.name"/>
     <callvar name = "pcdata.def" parms = "setSource ServiceOperationLocationToXML
setGenerator service.operation.location"/>
     <callvar name = "pcdata.def" parms = "setSource ServiceOperationInputToXML
setGenerator service.operation.input"/>
     <callvar name = "pcdata.def" parms = "setSource ServiceOperationOutputToXML
setGenerator service.operation.output"/>
    </parse>
  </element>
  </vardef>
</before>
<parse>
<callvar name = "ServiceSet.var"/>
<callvar name = "PrototypeService.var"/>
<callvar name = "Service.var"/>
<callvar name = "ServiceOperation.var"/>
</parse>
</tree>
```

XML to Java

```
<!DOCTYPE tree SYSTEM "tree.dtd">
<tree source = "null" pass-through = "false">
<before>
<vardef name = "business.var">
 <element source = "business"
       class = "com.veo.xdk.dev.schema.test.blib.Business"
       type = "7" setter = "setBusiness">
  <before>
   <onattribute name = "business.number">
    <actions>
     <callmeth name = "businessNumberFromXML">
      <parms>
       <getattr name = "business.number"/>
      </parms>
     </callmeth>
    </actions>
   </onattribute>
  </before>
```

```
    <parse>
     <callvar name = "party.name.var" parms = "setPosition -1"/>
     <callvar name = "address.set.var"/>
    </parse>
   </element>
  </vardef>
  <vardef name = "party.name.var">
   <element source = "party.name" setter = "partyNameFromXML" position = "-1" class =
  "java.lang.String">
    <parse>
     <data class = "java.lang.String" position = "0"/>
    </parse>
   </element>
  </vardef>
  <vardef name = "city.var">
   <element source = "city" setter = "cityFromXML" position = "-1" class = "java.lang.String">
    <parse>
     <data class = "java.lang.String" position = "0"/>
    </parse>
   </element>
  </vardef>
  <vardef name = "internet.var">
   <element source = "internet" setter = "internetFromXML" position = "-1" class =
  "java.lang.String">
    <parse>
     <data class = "java.lang.String" position = "0"/>
    </parse>
   </element>
  </vardef>
  <vardef name = "country.var">
   <element source = "country" setter = "countryFromXML" position = "-1" class =
  "java.lang.String">
    <parse>
     <data class = "java.lang.String" position = "0"/>
    </parse>
   </element>
  </vardef>
  <vardef name = "state.var">
   <element source = "state" setter = "stateFromXML" position = "-1" class = "java.lang.String">
    <parse>
     <data class = "java.lang.String" position = "0"/>
    </parse>
   </element>
  </vardef>
  <vardef name = "email.var">
   <element source = "email" setter = "emailFromXML" position = "-1" class =
  "java.lang.String">
    <parse>
     <data class = "java.lang.String" position = "0"/>
    </parse>
```

```
  </element>
 </vardef>
 <vardef name = "address.physical.var">
  <element source = "address.physical"
          class = "com.veo.xdk.dev.schema.test.blib.AddressPhysical"
          type = "7" setter = "setAddressPhysical">
   <before>
   </before>
   <parse>
    <callvar name = "street.var" parms = "setPosition -1"/>
    <callvar name = "city.var" parms = "setPosition -1"/>
    <callvar name = "state.var" parms = "setPosition -1"/>
    <callvar name = "postcode.var" parms = "setPosition -1"/>
    <callvar name = "country.var" parms = "setPosition -1"/>
   </parse>
  </element>
 </vardef>
 <vardef name = "telephone.var">
  <element source = "telephone" setter = "telephoneFromXML" position = "-1" class =
"java.lang.String">
   <parse>
    <data class = "java.lang.String" position = "0"/>
   </parse>
  </element>
 </vardef>
 <vardef name = "person.var">
  <element source = "person"
          class = "com.veo.xdk.dev.schema.test.blib.Person"
          type = "7" setter = "setPerson">
   <before>
    <onattribute name = "SSN">
     <actions>
      <callmeth name = "sSNFromXML">
       <parms>
        <getattr name = "SSN"/>
       </parms>
      </callmeth>
     </actions>
    </onattribute>
   </before>
   <parse>
    <callvar name = "party.name.var" parms = "setPosition -1"/>
    <callvar name = "address.set.var"/>
   </parse>
  </element>
 </vardef>
 <vardef name = "fax.var">
  <element source = "fax" setter = "faxFromXML" position = "-1" class = "java.lang.String">
   <parse>
    <data class = "java.lang.String" position = "0"/>
```

```
    </parse>
   </element>
  </vardef>
  <vardef name = "street.var">
   <element source = "street" setter = "streetFromXML" position = "-1" class =
  "java.lang.String">
    <parse>
     <data class = "java.lang.String" position = "0"/>
    </parse>
   </element>
  </vardef>
  <vardef name = "address.set.var">
   <element source = "address.set"
        class = "com.veo.xdk.dev.schema.test.blib.AddressSet"
        type = "7" setter = "setAddressSet">
    <before>
    </before>
    <parse>
     <callvar name = "address.physical.var"/>
     <callvar name = "telephone.var" parms = "setPosition -1"/>
     <callvar name = "fax.var" parms = "setPosition -1"/>
     <callvar name = "email.var" parms = "setPosition -1"/>
     <callvar name = "internet.var" parms = "setPosition -1"/>
    </parse>
   </element>
  </vardef>
  <vardef name = "postcode.var">
   <element source = "postcode" setter = "postcodeFromXML" position = "-1" class =
  "java.lang.String">
    <parse>
     <data class = "java.lang.String" position = "0"/>
    </parse>
   </element>
  </vardef>
  <vardef name = "market.participant.var">
   <element source = "market.participant"
        class = "com.veo.xdk.dev.schema.test.blib.MarketParticipant"
        type = "7" position = "0">
    <before>
    </before>
    <parse>
     <callvar name = "business.var"/>
     <callvar name = "person.var"/>
    </parse>
   </element>
  </vardef>
  </before>
  <parse>
  <callvar name = "business.var"/>
  <callvar name = "party.name.var"/>
```

```
<callvar name = "city.var"/>
<callvar name = "internet.var"/>
<callvar name = "country.var"/>
<callvar name = "state.var"/>
<callvar name = "email.var"/>
<callvar name = "address.physical.var"/>
<callvar name = "telephone.var"/>
<callvar name = "person.var"/>
<callvar name = "fax.var"/>
<callvar name = "street.var"/>
<callvar name = "address.set.var"/>
<callvar name = "postcode.var"/>
<callvar name = "market.participant.var"/>
</parse>
</tree>
```

Makefiles:

```
#
# this makefile was generated by bic version 0.0. 05/02/1998
#
#
#
#

# get the package name from the package argument passed to SchemaGen
PACKAGE_NAME = com/veo/xdk/dev/schema/test/blib

JAVA_SOURCES    +=          \
        MarketParticipant.java \
        Business.java \
        Person.java \
        Party.java \
        AddressPhysical.java \
        AddressSet.java \

MAKEFILE_MASTER_DIR = xxx
include $(MAKEFILE_MASTER_DIR)/Makefile.master

all:: $(JAVA_CLASSES)

#
# this makefile was generated by bic version 0.0. 05/02/1998
#
#
#
#

# get the package name from the package argument passed to SchemaGen
```

```
PACKAGE_NAME = com/veo/xdk/dev/schema/test/blib

JAVA_SOURCES   +=          \
        ServiceSet.java \
        PrototypeService.java \
        Service.java \
        ServiceOperation.java \

MAKEFILE_MASTER_DIR = xxx
include $(MAKEFILE_MASTER_DIR)/Makefile.master

all:: $(JAVA_CLASSES)
```

Finally, the XML document instances generated at run time according to the model above for one example follows:

```
<!DOCTYPE market.participant SYSTEM "market.participant.dtd" >
<market.participant>

<business business.number="1234567890" >

<party.name>IBM</party.name>

<address.set>
<address.physical>

<street>1 IBM Way</street>
<city>Palo Alto</city>
<state>CA</state>
<postcode>94304</postcode>
<country>USA</country>
</address.physical>

<telephone>123 456-7890</telephone>
<fax>123 456 0987</fax>
<email>ibmec@ibm.com</email>
</address.set>

</business>

</market.participant>

<!DOCTYPE service SYSTEM "service.dtd" >
<service.set>
<service>
<service.name>Order Service</service.name>
<service.location>www.ibm.com/order</service.location>
```

```
<service.operation>
<service.operation.name>Submit Order</service.operation.name>
<service.operation.location>www.ibm.com/order/submit</service.location>
<service.operation.input>urn:x-ibm:services:order:operations:po.dtd</service.operation.input>
<service.operation.output>urn:x-
ibm:services:order:operations:poack.dtd</service.operation.output>
</service.operation>

<service.operation>
<service.operation.name>Track Order</service.operation.name>
<service.operation.location>www.ibm.com/order/track</service.location>
<service.operation.input>urn:x-
ibm:services:order:operations:track.irequest.dtd</service.operation.input>
<service.operation.output>urn:x-
ibm:services:order:operations:track.iresponse.dtd</service.operation.output>
</service.operation>

</service>
</service.set>
```

Using the tools along with a BID composer application, which provides a drag, drop and forms editing user interface, a developer is able to create a business interface definition and to produce a well formed, valid business interface definition in the form of an XML document. Thus, the example run time instance is a business interface definition for an ordering service for IBM to be used by Ingram Micro and others to order laptop computers from IBM. (There is no relationship between the applicant and IBM or Ingram Micro). Utilizing these processes, a user is able to build a system that allows for programming of a business interface using the documents defined according to the present invention.

The role of CBL and the BID processor of the present invention in an XML/JAVA environment can be further understood by the following explanation of the processing of a Purchase Order.

Company A defines its Purchase Order document type using a visual programming environment that contains a library of CBL DTDs and modules, all defined using common business language elements so that they contain data type and other interpretation information. Company A's PO might just involve

minor customizations to a more generic "transaction document" specification that comes with the CBL library, or it might be built from the ground up from CBL modules for address, date and time, currency, etc.

The documentation for the generic "transaction document" specification (such as the transact.dtd set out above) typifies the manner in which CBL specifications are built from modules and are interlinked with other CBL DTDs.

A compiler takes the purchase order definition and generates several different target forms. All of these target forms can be derived through "tree to tree" transformations of the original specification. The most important for this example are:

(a) the XML DTD for the purchase order.

(b) a JAVA Bean that encapsulates the data structures for a purchase order (the JAVA classes, arguments, datatypes, methods, and exception structures are created that correspond to information in the Schema definition of the purchase order).

(c) A "marshaling" program that converts purchase orders that conform to the Purchase Order DTD into a Purchase Order JAVA Bean or loads them into a database, or creates HTML (or an XSL style sheet) for displaying purchase orders in a browser.

(d) An "unmarshaling" program that extracts the data values from Purchase Order JAVA Beans and converts them into an XML document that conforms to the Purchase Order DTD.

Now, back to the scenario. A purchasing application generates a Purchase Order that conforms to the DTD specified as the service interface for a supplier who accepts purchase orders.

The parser uses the purchase order DTD to decompose the purchase order instance into a stream of information about the elements and attribute values it contains. These "property sets" are then transformed into corresponding JAVA event objects by wrapping them with JAVA code. This

transformation in effect treats the pieces of marked-up XML document as instructions in a custom programming language whose grammar is defined by the DTD. These JAVA events can now be processed by the marshaling applications generated by the compiler to "load" JAVA Bean data structures.

Turning the XML document into a set of events for JAVA applications to process, is unlike the normal model of parsing in which the parser output is maintained as an internal data structure and processing does not begin until parsing completes. The event based processing, in response to the BID definitions, is the key to enabling the much richer functionality of the processor because it allows concurrent document application processing to begin as soon as the first event is emitted.

JAVA programs that "listen for" events of various types are generated from the Schema definition of those events. These listeners are programs created to carry out the business logic associated with the XML definitions in the CBL; for example, associated with an "address" element may be code that validates the postal code by checking a database. These listeners "subscribe" to events by registering with the document router, which directs the relevant events to all the subscribers who are interested in them.

This publish and subscribe architecture means that new listener programs can be added without knowledge by or impact on existing ones. Each listener has a queue into which the router directs its events, which enables multiple listeners can handle events in parallel at their own pace.

For the example purchase order here, there might be listeners for:

- the purchase order, which would connect it to an order entry program,
- product descriptions, which might check inventory,
- address information, which could check Fed Ex or other service for delivery availability,

- buyer information, which could check order history (for creditworthiness, or to offer a promotion, or similar processing based on knowing who the customer is).

Complex listeners can be created as configurations of primitive ones (e.g., a purchase order listener may contain and invoke these listeners here, or they may be invoked on their own).

Fig. 11 illustrates the market maker node in the network of Fig. 1. The market maker node includes the basic structures of the system of Fig. 3, including a network interface 1101, a document parser 1102, a document to host and host to document translator 1103, and a front end 1104, referred to as a router in this example. The market maker module 1105 in this example includes a set of business interface definitions, or other identifiers sufficient to support the market maker function, for participants in the market, a CBL repository, and a compiler all serving the participants in the market. The router 1104 includes a participant registry and document filters which respond to the events generated at the output of the translator and by the parser to route incoming documents according to the participant registry and according to the element and attribute filters amongst the listeners to the XML event generators. Thus, certain participants in the market may register to receive documents that meet prespecified parameters. For example, input documents according to a particular DTD, and including an attribute such as numbers of products to be purchased greater than a threshold, or such as a maximum price of a document request to be purchased, can be used to filter documents at the router 1104. Only such documents as match the information registered in the participant registry at the router 1104 are then passed on to the registered participant.

The router 1104 may also serve local host services 1105 and 1106, and as such act as a participant in the market as well as the market maker. Typically, documents that are received by the router 1104 are traversed to

determine the destinations to which such documents should be routed, there again passed back through the translator 1103, if necessary, and out the network interface 1101 to the respective destinations.

The market maker is a server that binds together a set of internal and external business services to create a virtual enterprise or trading community. The server parses incoming documents and invokes the appropriate services by, for example, handing off a request for product data to a catalog server or forwarding a purchase order to an ERP system. The server also handles translation tasks, mapping the information from a company's XML documents onto document formats used by trading partners and into data formats required by its legacy systems.

With respect to the service definition above, when a company submits a purchase order, the XML parser in the server uses the purchase order DTD to transform the purchase order instance into a stream of information events. These events are then routed to any application that is programmed to handle events of a given type; in some cases, the information is forwarded over the Internet to a different business entirely. In the purchase order example, several applications may act on information coming from the parser:

- An order entry program processes the purchase order as a complete message;
- An ERP system checks inventory for the products described in the purchase order;
- A customer database verifies or updates the customer's address;
- A shipping company uses the address information to schedule a delivery
- A bank uses the credit card information to authorize the transaction.

Trading partners need only agree on the structure, content, and sequencing of the business documents they exchange, not on the details of APIs. How a document is processed and what actions result is strictly up to the business providing the service. This elevates integration from the system level to the business level. It enables a business to present a clean and stable interface to its business partners despite changes in its internal technology implementation, organization, or processes.

Figs. 12, 13 and 14 illustrate processes executed at a market maker node in the system of Fig. 11. In Fig. 12, an input document is received at the network interface from an originating participant node (step 1200). The document is parsed (step 1201). The document is translated to the format of the host, for example XML to JAVA (step 1202). The host formatted events and objects are then passed to the router service (step 1203). The services registered to accept the document according to the document type and content of the document are identified (step 1204). The document or a portion of the document is passed to the identified services (step 1205). As service is performed in response to the document content (step 1206). The output data of the service is produced (step 1207). The output is converted to the document format, for example from a JAVA format to an XML format (step 1208). Finally, the output document is sent to a participant node (step 1209).

The registration service is one such function which is managed by the router. Thus, a market participant document is accepted at the network interface as shown in Fig. 13 (step 1300). The market participant document is stored in the business interface definition repository (step 1301) for the market maker node. In addition, the document is parsed (step 1302). The parsed document is translated into the format of the host (step 1303). Next, the document is passed to the router service (step 1304). The router service includes a listener which identifies the registration service as the destination of the document according to

the document type and content (step 1305). The document or elements of the document are passed to the registration service (step 1306). In the registration service, the needed service specifications are retrieved according to the business interface definition (step 1307). If the service specifications are gathered, at step 1308, the router service filters are set according to the business interface definition and the service specifications (step 1309). Registration acknowledgment data is produced (1310). The registration acknowledgment data is converted to a document format (step 1311). Finally, the acknowledgment document is sent to the participant node indicating to the participant that is successfully registered with the market maker (step 1312).

The process at step 1307 of gathering needed service specifications is illustrated for one example in Fig. 14. This process begins by locating a service business interface definition supported by the market participant (step 1400). The service definition is retrieved, for example by an E-mail transaction or web access to repository node (step 1401). The service specification is stored in the BID repository (step 1402). The service business interface definition document is parsed (step 1403). The parsed document is translated into the format of the host (step 1404). Host objects are passed to the router service (step 1405). The registration service is identified according to the document type and content (step 1406). Finally, the information in the service business interface definition document is passed to the registration service (step 1407) for use according to the process of Fig. 13.

Fig. 15 illustrates the processor, components and sequence of processing of incoming data at market maker node according to the present invention. The market maker node includes a communication agent 1500 at the network interface. The communication agent is coupled with an XML parser 1501 which supplies events to an XML processor 1502. The XML processor supplies events to a document router. The document router feeds a document service

1504 that provides an interface for supplying the received documents to the enterprise solution software 1505 in the host system. The communication agent 1500 is an Internet interface which includes appropriate protocol stacks supporting such protocols as HTTP, SMTP, FTP, or other protocols. Thus, the incoming data could come in an XML syntax, an ASCII data syntax or other syntax as suits a particular communication channel. All the documents received in non-XML syntaxes are translated into XML and passed the XML parser. A translation table 1506 is used to support the translation from non-XML form into XML form.

The converted documents are supplied to the parser 1501. The XML parser parses the received XML document according to the document type definition which matches it. If an error is found, then the parser sends the document back to the communication agent 1500. A business interface definition compiler BIDC 1507 acts as a compiler for business interface definition data. The DTD file for the XML parser, JAVA beans corresponding to the DTD file, and translation rules for translating DTD files to JAVA beans are created by compiling the BID data. An XML instance is translated to JAVA instance by referring to these tools. Thus the BID compiler 1507 stores the DTD documents 1508 and produces JAVA documents which correspond 1509. The XML documents are passed to the processor 1502 which translates them into the JAVA format. In a preferred system, JAVA documents which have the same status as the document type definitions received in the XML format are produced. The JAVA beans are passed to the document router 1503. The document router 1503 receives the JAVA beans and passes the received class to the appropriate document service using a registry program, for example using the event listener architecture described above. The document service 1504 which receives the document in the form of JAVA beans from the router 1503 acts as the interface to the enterprise solution software. This includes a registry

service 1510 by which listeners to XML events are coupled with the incoming data streams, and a service manager 1511 to manage the routing of the incoming documents to the appropriate services. The document service manager 1511 provides for administration of the registry service and for maintaining document consistency and the like.

The document service communicates with the back end system using any proprietary API, or using such more common forms as the CORBA/COM interface or other architectures.

Fig. 16 provides a heuristic diagram of the market maker and market participant structures according to the present invention. Thus, the electronic commerce market according to the present invention can be logically organized as set forth in Fig. 16. At the top of the organization, a market maker node 1600 is established. The market maker node includes resources that establish a marketplace 1601. Such resources include a market registry service and the like. Businesses 1602 register in the marketplace 1601 by publishing a business interface definition. The business interface definition defines the services 1603 for commercial transactions in which the businesses will participate. The transactions 1604 and services 1603 use documents 1605 to define the inputs and outputs, and outline the commercial relationship between participants in the transaction. The documents have content 1606 which carries the particulars of each transaction. The manner in which the content is processed by the participants in the market, and by the market maker is completely independent of the document based electronic commerce network which is established according to the present invention. Overall, a robust, scalable, intuitive structure is presented for enabling electronic commerce on communication networks is provided.

Thus, the present invention in an exemplary system provides a platform based on the XML processor and uses XML documents as the interface between

loosely coupled business systems. The documents are transferred between businesses and processed by participant nodes before entering the company business system. Thus the platform enables electronic commerce applications between businesses where each business system operates using different internal commerce platforms, processes and semantics, by specifying a common set of business documents and forms.

According to the present invention, virtual enterprises are created by interconnecting business systems and service, are primarily defined in terms of the documents (XML-encoded) that businesses accept and generate:

- "if you send me a request for a catalog, I will send you a catalog:
- "if you send me a purchase order and I can accept it, I will send you an invoice".

The foregoing description of a preferred embodiment of the invention has been presented for purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously, many modifications and variations will be apparent to practitioners skilled in this art. It is intended that the scope of the invention be defined by the following claims and their equivalents.

What is claimed is:

CLAIMS

1. An interface for transactions among nodes in a network including a plurality of nodes which execute processes involved in the transactions, comprising:

a machine readable specification of an interface to transaction processes stored in memory accessible by at least one node in the network, including interpretation information providing a definition of an input document, and a definition of an output document, the definitions of the input and output documents comprising respective descriptions of sets of storage units and logical structures for the sets of storage units.

2. The interface of claim 1, wherein the interpretation information includes data type specifications for at least one logical structure in the definitions of the input and output documents.

3. The interface of claim 1, wherein the interpretation information includes at least one data structure mapping predefined sets of storage units for a particular logical structure in the definitions of the input and output documents, to respective entries in a list.

4. The interface of claim 1, including a repository in memory accessible by at least one node in the network storing a library of logical structures, and interpretation information for logic structures.

5. The interface of claim 1, wherein the machine readable specification includes a document compliant with a definition of an interface document including logical structures for storing an identifier of a particular

transaction, and at least one of definitions and references to definitions of input and output documents for the particular transaction.

6. The interface of claim 1, wherein the machine readable specification includes a document compliant with a definition of an interface document including logical structures for storing an identifier of the interface, and for storing at least one of specifications and references to specifications of a set of one or more transactions supported by the interface.

7. The interface of claim 6, wherein the machine readable specification includes a reference to a specification of a particular transaction, and the specification of the particular transaction includes a document including logical structures for storing at least one of definitions and references to definitions of input and output documents for the particular transaction.

8. The interface of claim 1, wherein the storage units comprise parsed data.

9. The interface of claim 8, wherein the parsed data in at least one of the input and output documents comprises:

character data encoding text characters in the one of the input and output documents, and

markup data identifying sets of storage units according to the logical structure of the one of the input and output documents.

10. The interface of claim 9, wherein at least one of the sets of storage units encodes a plurality of text characters providing a natural language word.

11. The interface of claim 8, wherein the interpretation information for at least one of the sets of storage units identified by a particular logical structure of at least one of the input and output documents, encodes respective definitions for sets of parsed characters.

12. The interface of claim 8, wherein the storage units comprise unparsed data.

13. The interface of claim 1, including a repository stored in memory accessible by at least one node in the network of document types for use in a plurality of transactions, and wherein the definition of one of the input and output documents includes a reference to a document type in the repository.

14. The method of claim 13, wherein the repository of document types includes a document type for identifying participant processes in the network.

15. The interface of claim 1, wherein the definitions of the input and output documents comprise document type definitions compliant with a standard Extensible Markup Language XML.

16. The interface of claim 1, wherein the machine readable data structure including interpretation information comprises a document organized according to a document type definition compliant with a standard Extensible Markup Language XML.

17. Apparatus for establishing participant interfaces for transactions executed on a system including a network interface and a data processing resources which execute a transaction processes of the transactions according to a transaction processing architecture; comprising:

programs of instructions executable by the system, stored on a medium accessible by the system, providing tools to build a definition of a participant interface for a participant in a particular transaction, the definition of a participant interface including a definition of an input document for the participant and a definition of an output document for the participant, the definitions of the input and output documents comprising respective machine-readable descriptions of sets of storage units and logical structures for the sets of storage units; and

programs of instructions executable by the system, stored on a medium accessible by the system and responsive to the definitions of the input and output documents, to compile data structures corresponding to the sets of storage units and logical structures of the input and output documents compliant with the transaction processing architecture, to compile instructions executable by the system to translate the input document to the corresponding data structures, and to compile instructions executable by the system to translate output of the transaction processes into the sets of storage units and logical structures of the output document.

18. The apparatus of claim 17, wherein the tools to build a definition of a participant interface include instructions executable by the system to access elements of the definition from a repository, the repository storing a library of logical structures, and interpretation information for logic structures used to build interface descriptions.

19. The apparatus of claim 18, wherein the repository stores definitions of documents comprising logic structures.

20. The apparatus of claim 17, wherein the tools to build a definition of a participant interface include instructions executable by the system

to access a definition of another participant interface for a complementary transaction, the accessed definition including a definition of an input document for the complementary transaction, and a definition of an output document for the complementary transaction; and

to establish the definition of the participant interface by including the definition of the output document of the complementary transaction as the definition of the input document of the interface being built.

21. The apparatus of claim 20, wherein the tools to build a definition of a participant interface include instructions executable by the system

to include the definition of the input document of the complementary transaction as the definition of the output document of interface being built.

22. The apparatus of claim 17, wherein the tools to build a definition of a participant interface include instructions executable by the system to build a document compliant with a definition of a participant interface document including logical structures for storing an identifier of a particular transaction, and at least one of definitions and references to definitions of input and output documents for the particular transaction.

23. The apparatus of claim 17, wherein the tools to build a definition of a participant interface include instructions executable by the system to build a document compliant with a definition of a participant interface document including logical structures for storing an identifier of the participant interface, and for storing at least one of specifications and references to specifications of a set of one or more transactions supported by the participant interface.

24. The apparatus of claim 23, wherein the a document compliant with a definition of a participant interface document includes a reference to a machine-readable specification of a particular transaction, and the specification of the particular transaction includes a document including logical structures for storing at least one of definitions and references to definitions of input and output documents for the particular transaction.

25. The apparatus of claim 17, wherein the storage units comprise parsed data.

26. The apparatus of claim 25, wherein the parsed data in at least one of the input and output documents comprises:

character data encoding text characters in the one of the input and output documents, and

markup data identifying sets of storage units according to the logical structure of the one of the input and output documents.

27. The apparatus of claim 26, wherein at least one of the sets of storage units encodes a plurality of text characters providing a natural language word.

28. The apparatus of claim 25, wherein the specification includes interpretation information for at least one of the sets of storage units identified by the logical structure of at least one of the input and output documents, encoding respective definitions for sets of parsed characters.

29. The apparatus of claim 25, wherein the storage units comprise unparsed data.

30. The apparatus of claim 17, wherein data structures corresponding to the sets of storage units and logical structures of the input and output documents include programming objects including variables and methods according to the variant transaction processing architecture.

31. The apparatus of claim 17, wherein the variant transaction processing architectures of the transaction process includes comprises a process compliant with an interface description language.

32. The apparatus of claim 17, wherein the definitions of the input and output documents comprise document type definitions compliant with a standard Extensible Markup Language XML.

33. The apparatus of claim 19, wherein the definition of one of the input and output documents includes a reference to a document type in the repository.

34. The apparatus of claim 18, wherein the repository includes interpretation information specifying measurements of products subject of transactions.

35. The apparatus of claim 18, wherein the repository includes interpretation information specifying costs of products subject of transactions.

36. The apparatus of claim 18, wherein the repository includes interpretation information specifying features of products subject of transactions.

37. The apparatus of claim 18, wherein the repository includes interpretation information specifying financial terms of transactions.

38. The apparatus of claim 18, wherein the repository includes interpretation information specifying terms of shipment for products subject of transactions.

39. Apparatus for establishing participant interfaces for transactions executed on a system; comprising:

memory storing data and programs of instructions;

a data processor coupled to the memory which executes the programs of instructions; wherein the programs of instructions include

tools to build a definition of a participant interface for a participant in a particular transaction, the definition of a participant interface including a definition of an input document for the participant and a definition of an output document for the participant, the definitions of the input and output documents comprising

respective machine-readable descriptions of sets of storage units and logical structures for the sets of storage units; and

a compiler, responsive to the definitions of the input and output documents, to compile data structures corresponding to the sets of storage units and logical structures of the input and output documents compliant with the transaction processing architecture, to compile instructions executable by the system to translate the input document to the corresponding data structures, and to compile instructions executable by the system to translate output of the transaction processes into the sets of storage units and logical structures of the output document.

40. The apparatus of claim 39, including a repository stored in memory accessible by the data processor, and wherein the tools to build a definition of a participant interface include instructions executable by the system to access elements of the definition from the repository, the repository storing a library of logical structures, and interpretation information for logic structures used to build interface descriptions.

41. The apparatus of claim 40, wherein the repository stores definitions of documents comprising logic structures.

42. The apparatus of claim 39, wherein the tools to build a definition of a participant interface include instructions executable by the system

to access a definition of another participant interface for a complementary transaction, the accessed definition including a definition of an

input document for the complementary transaction, and a definition of an output document for the complementary transaction; and

to establish the definition of the participant interface by including the definition of the output document of the complementary transaction as the definition of the input document of the interface being built.

43. The apparatus of claim 42, wherein the tools to build a definition of a participant interface include instructions executable by the system

to include the definition of the input document of the complementary transaction as the definition of the output document of interface being built.

44. The apparatus of claim 39, wherein the tools to build a definition of a participant interface include instructions executable by the system to build a document compliant with a definition of a participant interface document including logical structures for storing an identifier of a particular transaction, and at least one of definitions and references to definitions of input and output documents for the particular transaction.

45. The apparatus of claim 39, wherein the tools to build a definition of a participant interface include instructions executable by the system to build a document compliant with a definition of a participant interface document including logical structures for storing an identifier of the participant interface, and for storing at least one of specifications and references to specifications of a set of one or more transactions supported by the participant interface.

46. The apparatus of claim 45, wherein the a document compliant with a definition of a participant interface document includes a reference to a machine-readable specification of a particular transaction, and the specification of the particular transaction includes a document including logical structures for storing at least one of definitions and references to definitions of input and output documents for the particular transaction.

47. The apparatus of claim 39, wherein the storage units comprise parsed data.

48. The apparatus of claim 47, wherein the parsed data in at least one of the input and output documents comprises:

character data encoding text characters in the one of the input and output documents, and

markup data identifying sets of storage units according to the logical structure of the one of the input and output documents.

49. The apparatus of claim 48, wherein at least one of the sets of storage units encodes a plurality of text characters providing a natural language word.

50. The apparatus of claim 47, wherein the specification includes interpretation information for at least one of the sets of storage units identified by the logical structure of at least one of the input and output documents, encoding respective definitions for sets of parsed characters.

51. The apparatus of claim 47, wherein the storage units comprise unparsed data.

52. The apparatus of claim 39, wherein data structures corresponding to the sets of storage units and logical structures of the input and output documents include programming objects including variables and methods according to the variant transaction processing architecture.

53. The apparatus of claim 39, wherein the variant transaction processing architectures of the transaction process comprises a process compliant with an interface description language.

54. The apparatus of claim 39, wherein the definitions of the input and output documents comprise document type definitions compliant with a standard Extensible Markup Language XML.

55. The apparatus of claim 41, wherein the definition of one of the input and output documents includes a reference to a document type in the repository.

56. The apparatus of claim 40, wherein the repository includes interpretation information specifying measurements of products subject of transactions.

57. The apparatus of claim 40, wherein the repository includes interpretation information specifying costs of products subject of transactions.

58. The apparatus of claim 40, wherein the repository includes interpretation information specifying features of products subject of transactions.

59. The apparatus of claim 40, wherein the repository includes interpretation information specifying financial terms of transactions.

60. The apparatus of claim 40, wherein the repository includes interpretation information specifying terms of shipment for products subject of transactions.

61. A method for programming a commercial transaction in a network, comprising:

defining a machine readable definition of an input document for a node in the network including resources to execute a process in the transaction, and a machine readable definition of an output document for the node, the definitions of the input and output documents comprising respective descriptions of sets of storage units and logical structures for the sets of storage units; and

providing interpretation information for the logical structures to the node.

62. The method of claim 61, wherein the interpretation information includes data type specifications for at least one logical structure in the definitions of the input and output documents.

63. The method of claim 61, wherein the interpretation information includes at least one data structure mapping predefined sets of storage units for a particular logical structure in the definitions of the input and output documents, to respective entries in a list.

64. The method of claim 61, the step of providing interpretation information includes providing a repository in memory accessible by at least one node in the network storing a library of logical structures, and interpretation information for logic structures.

65. The method of claim 61, including defining a machine readable specification of an interface including a document compliant with a definition of an interface document including logical structures for storing an identifier of a particular transaction, and at least one of the definitions and references to the definitions of the input and output document.

66. The method of claim 61, wherein the storage units comprise parsed data.

67. The method of claim 66, wherein the parsed data in at least one of the input and output documents comprises:

character data encoding text characters in the one of the input and output documents, and

markup data identifying sets of storage units according to the logical structure of the one of the input and output documents.

68. The method of claim 67, wherein at least one of the sets of storage units encodes a plurality of text characters providing a natural language word.

69. The method of claim 67, wherein the interpretation information for at least one of the sets of storage units identified by a particular logical

structure of at least one of the input and output documents, encodes respective definitions for sets of parsed characters.

70. The method of claim 66, wherein the storage units comprise unparsed data.

71. The method of claim 61, wherein the definitions of the input and output documents comprise document type definitions compliant with a standard Extensible Markup Language XML.

72. The method of claim 61, including:

providing a parser to generate event signals in response to logical structures in the definition of the input document; and

providing event listener programs which respond to the event signals to execute the process.

DOCUMENTS FOR COMMERCE IN TRADING PARTNER NETWORKS AND INTERFACE DEFINITIONS BASED ON THE DOCUMENTS

ABSTRACT

Machine readable documents connect businesses with customers, suppliers and trading partners. The self defining electronic documents, such as XML based documents, can be easily understood amongst the partners. Definitions of these electronics business documents, called business interface definitions, are posted on the Internet, or otherwise communicated to members of the network. The business interface definitions tell potential trading partners the services the company offers and the documents to use when communicating with such services. Thus, a typical business interface definition allows a customer to place an order by submitting a purchase order or a supplier checks availability by downloading an inventory status report. Also, composition of the input and output documents, coupled with interpretation information in a common business library, programs the transaction in a way which closely parallels the way in which paper based businesses operate.

II. Evidence Relied Upon By the Examiner

McKendrick Reference

McKendrick, Banks begin to play with XML, Bank Technology News, Sep. 1998, Vol. 11, Iss. 9, pg. 6, 2 pgs.

XML 1.0 Reference

W3C, *Extensible Markup Language* (XML) 1.0, 2/10/98, pages 1-37.

McKendrick Reference. ***McKendrick, Banks begin to play with XML***, Bank Technology News, Sep. 1998, Vol. 11, Iss. 9, pg. 6, 2 pgs.

Banks begin to play with XML

Joseph McKendrick. **Bank Technology News**. New York: Sep 1998. Vol. 11, Iss. 9; pg. 6, 2 pgs

» Jump to full text

Author(s):	Joseph McKendrick
Publication title:	Bank Technology News. New York: Sep 1998. Vol. 11, Iss. 9; pg. 6, 2 pgs
Source Type:	Periodical
ISSN/ISBN:	10603506
ProQuest document ID:	34406235
Text Word Count	645
Article URL:	http://gateway.proquest.com/openurl?ctx_ver=z39.88-2003&res_id=xri:pqd&rft_val_fmt=ori:fmt:kev:mtx:journal&genre=article&rft_id=xri:pqd:did=00000003

Full Text (645 words)

Financial institutions are tinkering with Extensible Markup Language (XML), a Web lingo unveiled last year. ⓘAmerican Century Investments, Discover Brokerage Direct and ⓘT. Rowe Price have upgraded to the latest version of Innovision's Financial Server, which uses XML. And Citibank is also trying out XML flavored electronic data interchange (EDI).

XML can enhance the quest for data on banks' Web sites. If a Web site contains XML, customers could click on a product listing and instantly view a more detailed description than they would on a site lacking XML. A CD offer on an XML site, for example, may feature more up-to-theminute interest rates, and in greater detail, than on a site without XML. And a customer query on that CD could be sorted out much more rapidly with XML. This is because XML can snag bigger chunks of information from back-end databases and quickly pop it into spontaneously crafted Web pages.

These benefits are making it easier for ⓘT. Rowe Price to fetch customer data and shoot it out to end users, drumming up big cost savings, says Kirk Kness, vice president of technology at the firm. The Baltimorebased firm is using the Innovision XML server to support ⓘT. Rowe's Online Investment Tracking service, which uses Quicken 98 as a front end.

On the market

ⓘT. Rowe Price is working with the one home banking vendor that's already melded XML into its products. Innovision Corp. recently began shipping its latest Open Financial Exchange (OFX)-based Financial Server product, capable of moving XMLbased protocols (See "OFX, Straight From The Heartland," December 1997 BTN). With both OFX and Open Buying On The Internet (OBI) striving to integrate XML into their respective electronic commerce standards, Innovision is certainly ahead of the curve.

"There's a lot of buzz-and confusion- about XML," quips Bill Cary, president of Innovision, Lexena, KS. "XML is a

h e c eb e ch e

technique or tool-you don't go out and buy XML. While 99 percent of the buzz about XML is related to Web documents and browsers, the real key is how XML can be applied to develop better data interchange standards and distributed universal applications."

Innovision's XML protocol server lets financial institutions "leverage their existing systems to connect directly with their accountholders," Cary says.

Innovision's customers agree with Cary's sentiment. The new product version "is allowing us to complete our vision for next-generation technology," says John MacIlwaine, chief technology officer, Discover Brokerage Direct, San Francisco.

Enlarge 200%
Enlarge 400%

User Friendly

Some experts believe XML will prove to be a boon for e-commerce applications. "XML will have a dramatic impact on the future development and growth of electronic commerce," says Ronald Whiting, president and CEO of CommerceNet, Washington, DC. Whiting believes that XML will allow companies to cooperate on product design and development, and better sync up with business partners.

EDI to boon

Case in point is Citibank's pilot jof **XML** powered EDI. The test-under the aegis of the Financial Services Technology Consortium's Bank **Internet** Payment System-is exploring the use of the new lingo for business-to-business **transactions**. The $311 billion (pre-merger) Citibank is also participating in other standards groups **working** with the new protocols. These efforts include the World Wide Web Consortium's subcommittee on **XML**, the folks who drafted this new **Internet** verbiage in the first place. Citibank's Vice President of External Standards Dan Schutzer, notes, "We are very excited at the prospect of **XML** for the next generation of Web documents, supporting a simpler means for communication."

As such, XML may be just the ticket for providing better customer service. "Customer services are now migrating to Web sites from call centers and physical locations," states a report from Microsoft Corp. "And, because most of **these business applications involve manipulation and transfer of data-such as purchase orders, invoices, customer information and appointments XML will allow a rich array of business applications to be implemented.**"

[Author Affiliation]
Joseph McKendrick is a technology writer based in Doylestown, PA.

h e c eb e ch e

XML 1.0 Reference. W3C, *Extensible Markup Language* (XML) 1.0, 2/10/98, pages 1-37.

REC-xml-19980210

W3C

Extensible Markup Language (XML) 1.0

W3C Recommendation 10-February-1998

This version:
http://www.w3.org/TR/1998/REC-xml-19980210
http://www.w3.org/TR/1998/REC-xml-19980210.xml
http://www.w3.org/TR/1998/REC-xml-19980210.html
http://www.w3.org/TR/1998/REC-xml-19980210.pdf
http://www.w3.org/TR/1998/REC-xml-19980210.ps
Latest version:
http://www.w3.org/TR/REC-xml
Previous version:
http://www.w3.org/TR/PR-xml-971208
Editors:
Tim Bray (Textuality and Netscape) <tbray@textuality.com>
Jean Paoli (Microsoft) <jeanpa@microsoft.com>
C. M. Sperberg-McQueen (University of Illinois at Chicago) <cmsmcq@uic.edu>

Abstract

The Extensible Markup Language (XML) is a subset of SGML that is completely described in this document. Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

Status of this document

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

This document specifies a syntax created by subsetting an existing, widely used international text processing standard (Standard Generalized Markup Language, ISO 8879:1986(E) as amended and corrected) for use on the World Wide Web. It is a product of the W3C XML Activity, details of which can be found at http://www.w3.org/XML. A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR.

This specification uses the term URI, which is defined by [Berners-Lee et al.], a work in progress expected to update [IETF RFC1738] and [IETF RFC1808].

The list of known errors in this specification is available at http://www.w3.org/XML/xml-19980210-errata.

Please report errors in this document to xml-editor@w3.org.

Extensible Markup Language (XML) 1.0

Table of Contents

R-195

Table of Contents

Appendices

R-196

1. Introduction

Extensible Markup Language, abbreviated XML, describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [ISO 8879]. By construction, XML documents are conforming SGML documents.

XML documents are made up of storage units called entities, which contain either parsed or unparsed data. Parsed data is made up of characters, some of which form character data, and some of which form markup. Markup encodes a description of the document's storage layout and logical structure. XML provides a mechanism to impose constraints on the storage layout and logical structure.

A software module called an **XML processor** is used to read XML documents and provide access to their content and structure. It is assumed that an XML processor is doing its work on behalf of another module, called the **application**. This specification describes the required behavior of an XML processor in terms of how it must read XML data and the information it must provide to the application.

1.1 Origin and Goals

XML was developed by an XML Working Group (originally known as the SGML Editorial Review Board) formed under the auspices of the World Wide Web Consortium (W3C) in 1996. It was chaired by Jon Bosak of Sun Microsystems with the active participation of an XML Special Interest Group (previously known as the SGML Working Group) also organized by the W3C. The membership of the XML Working Group is given in an appendix. Dan Connolly served as the WG's contact with the W3C.

The design goals for XML are:

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

This specification, together with associated standards (Unicode and ISO/IEC 10646 for characters, Internet RFC 1766 for language identification tags, ISO 639 for language name codes, and ISO 3166 for country name codes), provides all the information necessary to understand XML Version 1.0 and construct computer programs to process it.

This version of the XML specification may be distributed freely, as long as all text and legal notices remain intact.

R-197

1.2 Terminology

The terminology used to describe XML documents is defined in the body of this specification. The terms defined in the following list are used in building those definitions and in describing the actions of an XML processor:

may
Conforming documents and XML processors are permitted to but need not behave as described.

must
Conforming documents and XML processors are required to behave as described; otherwise they are in error.

error
A violation of the rules of this specification; results are undefined. Conforming software may detect and report an error and may recover from it.

fatal error
An error which a conforming XML processor must detect and report to the application. After encountering a fatal error, the processor may continue processing the data to search for further errors and may report such errors to the application. In order to support correction of errors, the processor may make unprocessed data from the document (with intermingled character data and markup) available to the application. Once a fatal error is detected, however, the processor must not continue normal processing (i.e., it must not continue to pass character data and information about the document's logical structure to the application in the normal way).

at user option
Conforming software may or must (depending on the modal verb in the sentence) behave as described; if it does, it must provide users a means to enable or disable the behavior described.

validity constraint
A rule which applies to all valid XML documents. Violations of validity constraints are errors; they must, at user option, be reported by validating XML processors.

well-formedness constraint
A rule which applies to all well-formed XML documents. Violations of well-formedness constraints are fatal errors.

match
(Of strings or names:) Two strings or names being compared must be identical. Characters with multiple possible representations in ISO/IEC 10646 (e.g. characters with both precomposed and base+diacritic forms) match only if they have the same representation in both strings. At user option, processors may normalize such characters to some canonical form. No case folding is performed. (Of strings and rules in the grammar:) A string matches a grammatical production if it belongs to the language generated by that production. (Of content and content models:) An element matches its declaration when it conforms in the fashion described in the constraint "Element Valid".

for compatibility
A feature of XML included solely to ensure that XML remains compatible with SGML.

for interoperability
A non-binding recommendation included to increase the chances that XML documents can be processed by the existing installed base of SGML processors which predate the WebSGML Adaptations Annex to ISO 8879.

2. Documents

A data object is an **XML document** if it is well-formed, as defined in this specification. A well-formed XML document may in addition be valid if it meets certain further constraints.

Each XML document has both a logical and a physical structure. Physically, the document is composed of units called entities. An entity may refer to other entities to cause their inclusion in the document. A document begins in a "root" or document entity. Logically, the document is composed of declarations,

R-198

elements, comments, character references, and processing instructions, all of which are indicated in the document by explicit markup. The logical and physical structures must nest properly, as described in "4.3.2 Well-Formed Parsed Entities".

2.1 Well-Formed XML Documents

A textual object is a well-formed XML document if:

1. Taken as a whole, it matches the production labeled `document`.
2. It meets all the well-formedness constraints given in this specification.
3. Each of the parsed entities which is referenced directly or indirectly within the document is well-formed.

Document

```
[1] document ::= prolog element Misc*
```

Matching the `document` production implies that:

1. It contains one or more elements.
2. There is exactly one element, called the **root**, or document element, no part of which appears in the content of any other element. For all other elements, if the start-tag is in the content of another element, the end-tag is in the content of the same element. More simply stated, the elements, delimited by start- and end-tags, nest properly within each other.

As a consequence of this, for each non-root element `C` in the document, there is one other element `P` in the document such that `C` is in the content of `P`, but is not in the content of any other element that is in the content of `P`. `P` is referred to as the **parent** of `C`, and `C` as a **child** of `P`.

2.2 Characters

A parsed entity contains **text**, a sequence of characters, which may represent markup or character data. A **character** is an atomic unit of text as specified by ISO/IEC 10646 [ISO/IEC 10646]. Legal characters are tab, carriage return, line feed, and the legal graphic characters of Unicode and ISO/IEC 10646. The use of "compatibility characters", as defined in section 6.8 of [Unicode], is discouraged.

Character Range

```
[2] Char ::= #x9 | #xA | #xD | [#x20-#xD7FF]          /* any Unicode character,
                | [#xE000-#xFFFD]                       excluding the surrogate
                | [#x10000-#x10FFFF]                    blocks, FFFE, and FFFF. */
```

The mechanism for encoding character code points into bit patterns may vary from entity to entity. All XML processors must accept the UTF-8 and UTF-16 encodings of 10646; the mechanisms for signaling which of the two is in use, or for bringing other encodings into play, are discussed later, in "4.3.3 Character Encoding in Entities".

2.3 Common Syntactic Constructs

This section defines some symbols used widely in the grammar.

`S` (white space) consists of one or more space (#x20) characters, carriage returns, line feeds, or tabs.

White Space

```
[3] S ::= (#x20 | #x9 | #xD | #xA)+
```

Characters are classified for convenience as letters, digits, or other characters. Letters consist of an alphabetic or syllabic base character possibly followed by one or more combining characters, or of an ideographic character. Full definitions of the specific characters in each class are given in "B. Character Classes".

A **Name** is a token beginning with a letter or one of a few punctuation characters, and continuing with letters, digits, hyphens, underscores, colons, or full stops, together known as name characters. Names beginning with the string "`xml`", or any string which would match `(('X'|'x') ('M'|'m') ('L'|'l'))`, are reserved for standardization in this or future versions of this specification.

Note: The colon character within XML names is reserved for experimentation with name spaces. Its meaning is expected to be standardized at some future point, at which point those documents using the colon for experimental purposes may need to be updated. (There is no guarantee that any name-space mechanism adopted for XML will in fact use the colon as a name-space delimiter.) In practice, this means that authors should not use the colon in XML names except as part of name-space experiments, but that XML processors should accept the colon as a name character.

An `Nmtoken` (name token) is any mixture of name characters.

Names and Tokens

```
[4] NameChar ::= Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar
                 | Extender
[5]      Name ::= (Letter | '_' | ':') (NameChar)*
[6]     Names ::= Name (S Name)*
[7]  Nmtoken ::= (NameChar)+
[8] Nmtokens ::= Nmtoken (S Nmtoken)*
```

Literal data is any quoted string not containing the quotation mark used as a delimiter for that string. Literals are used for specifying the content of internal entities (`EntityValue`), the values of attributes (`AttValue`), and external identifiers (`SystemLiteral`). Note that a `SystemLiteral` can be parsed without scanning for markup.

Literals

```
 [9]   EntityValue ::= '"' ([^%&"] | PEReference | Reference)* '"'
                     |  "'" ([^%&'] | PEReference | Reference)* "'"
[10]      AttValue ::= '"' ([^<&"] | Reference)* '"'
                     |  "'" ([^<&'] | Reference)* "'"
[11] SystemLiteral ::= ('"' [^"]* '"') | ("'" [^']* "'")
[12]  PubidLiteral ::= '"' PubidChar* '"' | "'" (PubidChar - "'")* "'"
[13]    PubidChar ::= #x20 | #xD | #xA | [a-zA-Z0-9] | [-'()+,./:=?;!*#@$_%]
```

2.4 Character Data and Markup

Text consists of intermingled character data and markup. **Markup** takes the form of start-tags, end-tags,

R-200

empty-element tags, entity references, character references, comments, CDATA section delimiters, document type declarations, and processing instructions.

All text that is not markup constitutes the **character data** of the document.

The ampersand character (&) and the left angle bracket (<) may appear in their literal form *only* when used as markup delimiters, or within a comment, a processing instruction, or a CDATA section. They are also legal within the literal entity value of an internal entity declaration; see "4.3.2 Well-Formed Parsed Entities". If they are needed elsewhere, they must be escaped using either numeric character references or the strings "`&`" and "`<`" respectively. The right angle bracket (>) may be represented using the string "`>`", and must, for compatibility, be escaped using "`>`" or a character reference when it appears in the string "`]]>`" in content, when that string is not marking the end of a CDATA section.

In the content of elements, character data is any string of characters which does not contain the start-delimiter of any markup. In a CDATA section, character data is any string of characters not including the CDATA-section-close delimiter, "`]]>`".

To allow attribute values to contain both single and double quotes, the apostrophe or single-quote character (') may be represented as "`&apos;`", and the double-quote character (") as "`&quot;`".

Character Data

```
[14] CharData ::= [^<&]* - ([^<&]* ']]>' [^<&]*)
```

2.5 Comments

Comments may appear anywhere in a document outside other markup; in addition, they may appear within the document type declaration at places allowed by the grammar. They are not part of the document's character data; an XML processor may, but need not, make it possible for an application to retrieve the text of comments. For compatibility, the string "--" (double-hyphen) must not occur within comments.

Comments

```
[15] Comment ::= '<!--' ((Char - '-') | ('-' (Char - '-')))* '-->'
```

An example of a comment:

```
<!-- declarations for <head> & <body> -->
```

2.6 Processing Instructions

Processing instructions (PIs) allow documents to contain instructions for applications.

Processing Instructions

```
[16]       PI ::= '<?' PITarget (S (Char* - (Char* '?>' Char*)))? '?>'
[17] PITarget ::= Name - (('X' | 'x') ('M' | 'm') ('L' | 'l'))
```

PIs are not part of the document's character data, but must be passed through to the application. The PI

R-201

begins with a target (PITarget) used to identify the application to which the instruction is directed. The target names "XML", "xml", and so on are reserved for standardization in this or future versions of this specification. The XML Notation mechanism may be used for formal declaration of PI targets.

2.7 CDATA Sections

CDATA sections may occur anywhere character data may occur; they are used to escape blocks of text containing characters which would otherwise be recognized as markup. CDATA sections begin with the string "<![CDATA[" and end with the string "]]>":

CDATA Sections

```
[18] CDSect ::= CDStart CData CDEnd
[19] CDStart ::= '<![CDATA['
[20] CData ::= (Char* - (Char* ']]>' Char*))
[21] CDEnd ::= ']]>'
```

Within a CDATA section, only the CDEnd string is recognized as markup, so that left angle brackets and ampersands may occur in their literal form; they need not (and cannot) be escaped using "<" and "&". CDATA sections cannot nest.

An example of a CDATA section, in which "<greeting>" and "</greeting>" are recognized as character data, not markup:

```
<![CDATA[<greeting>Hello, world!</greeting>]]>
```

2.8 Prolog and Document Type Declaration

XML documents may, and should, begin with an **XML declaration** which specifies the version of XML being used. For example, the following is a complete XML document, well-formed but not valid:

```
<?xml version="1.0"?>
<greeting>Hello, world!</greeting>
```

and so is this:

```
<greeting>Hello, world!</greeting>
```

The version number "1.0" should be used to indicate conformance to this version of this specification; it is an error for a document to use the value "1.0" if it does not conform to this version of this specification. It is the intent of the XML working group to give later versions of this specification numbers other than "1.0", but this intent does not indicate a commitment to produce any future versions of XML, nor if any are produced, to use any particular numbering scheme. Since future versions are not ruled out, this construct is provided as a means to allow the possibility of automatic version recognition, should it become necessary. Processors may signal an error if they receive documents labeled with versions they do not support.

The function of the markup in an XML document is to describe its storage and logical structure and to associate attribute-value pairs with its logical structures. XML provides a mechanism, the document type declaration, to define constraints on the logical structure and to support the use of predefined storage units. An XML document is **valid** if it has an associated document type declaration and if the document

R-202

complies with the constraints expressed in it.

The document type declaration must appear before the first element in the document.

Prolog

```
[22]      prolog ::= XMLDecl? Misc* (doctypedecl Misc*)?
[23]    XMLDecl  ::= '<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>'
[24] VersionInfo ::= S 'version' Eq (' VersionNum ' | " VersionNum ")
[25]          Eq ::= S? '=' S?
[26] VersionNum  ::= ([a-zA-Z0-9_.:] | '-')+
[27]        Misc ::= Comment | PI | S
```

The XML **document type declaration** contains or points to markup declarations that provide a grammar for a class of documents. This grammar is known as a document type definition, or **DTD**. The document type declaration can point to an external subset (a special kind of external entity) containing markup declarations, or can contain the markup declarations directly in an internal subset, or can do both. The DTD for a document consists of both subsets taken together.

A **markup declaration** is an element type declaration, an attribute-list declaration, an entity declaration, or a notation declaration. These declarations may be contained in whole or in part within parameter entities, as described in the well-formedness and validity constraints below. For fuller information, see "4. Physical Structures".

Document Type Definition

```
[28] doctypedecl ::= '<!DOCTYPE' S Name (S ExternalID)? S? ('[' (markupdecl | PEReference | S)* ']' S?)? '>'   [ VC: Root Element Type ]
[29] markupdecl  ::= elementdecl | AttlistDecl | EntityDecl | NotationDecl | PI | Comment                      [ VC: Proper Declaration/PE Nesting ]
                                                                                                               [ WFC: PEs in Internal Subset ]
```

The markup declarations may be made up in whole or in part of the replacement text of parameter entities. The productions later in this specification for individual nonterminals (elementdecl, AttlistDecl, and so on) describe the declarations *after* all the parameter entities have been included.

Validity Constraint: Root Element Type
The Name in the document type declaration must match the element type of the root element.

Validity Constraint: Proper Declaration/PE Nesting
Parameter-entity replacement text must be properly nested with markup declarations. That is to say, if either the first character or the last character of a markup declaration (markupdecl above) is contained in the replacement text for a parameter-entity reference, both must be contained in the same replacement text.

Well-Formedness Constraint: PEs in Internal Subset
In the internal DTD subset, parameter-entity references can occur only where markup declarations can occur, not within markup declarations. (This does not apply to references that occur in external parameter entities or to the external subset.)

R-203

Like the internal subset, the external subset and any external parameter entities referred to in the DTD must consist of a series of complete markup declarations of the types allowed by the non-terminal symbol markupdecl, interspersed with white space or parameter-entity references. However, portions of the contents of the external subset or of external parameter entities may conditionally be ignored by using the conditional section construct; this is not allowed in the internal subset.

External Subset

```
[30]     extSubset ::= TextDecl? extSubsetDecl
[31] extSubsetDecl ::= ( markupdecl | conditionalSect | PEReference | S )*
```

The external subset and external parameter entities also differ from the internal subset in that in them, parameter-entity references are permitted *within* markup declarations, not only *between* markup declarations.

An example of an XML document with a document type declaration:

```
<?xml version="1.0"?>
<!DOCTYPE greeting SYSTEM "hello.dtd">
<greeting>Hello, world!</greeting>
```

The system identifier "hello.dtd" gives the URI of a DTD for the document.

The declarations can also be given locally, as in this example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE greeting [
  <!ELEMENT greeting (#PCDATA)>
]>
<greeting>Hello, world!</greeting>
```

If both the external and internal subsets are used, the internal subset is considered to occur before the external subset. This has the effect that entity and attribute-list declarations in the internal subset take precedence over those in the external subset.

2.9 Standalone Document Declaration

Markup declarations can affect the content of the document, as passed from an XML processor to an application; examples are attribute defaults and entity declarations. The standalone document declaration, which may appear as a component of the XML declaration, signals whether or not there are such declarations which appear external to the document entity.

Standalone Document Declaration

```
[32] SDDecl ::= S 'standalone' Eq (("'" ('yes' | 'no') "'") | ('"' ('yes' | 'no') '"'))   [ VC: Standalone Document Declaration ]
```

In a standalone document declaration, the value "yes" indicates that there are no markup declarations external to the document entity (either in the DTD external subset, or in an external parameter entity referenced from the internal subset) which affect the information passed from the XML processor to the application. The value "no" indicates that there are or may be such external markup declarations. Note that the standalone document declaration only denotes the presence of external *declarations*; the

R-204

presence, in a document, of references to external *entities*, when those entities are internally declared, does not change its standalone status.

If there are no external markup declarations, the standalone document declaration has no meaning. If there are external markup declarations but there is no standalone document declaration, the value "no" is assumed.

Any XML document for which `standalone="no"` holds can be converted algorithmically to a standalone document, which may be desirable for some network delivery applications.

Validity Constraint: Standalone Document Declaration
The standalone document declaration must have the value "no" if any external markup declarations contain declarations of:

- attributes with default values, if elements to which these attributes apply appear in the document without specifications of values for these attributes, or
- entities (other than `amp`, `lt`, `gt`, `apos`, `quot`), if references to those entities appear in the document, or
- attributes with values subject to normalization, where the attribute appears in the document with a value which will change as a result of normalization, or
- element types with element content, if white space occurs directly within any instance of those types.

An example XML declaration with a standalone document declaration:

```
<?xml version="1.0" standalone='yes'?>
```

2.10 White Space Handling

In editing XML documents, it is often convenient to use "white space" (spaces, tabs, and blank lines, denoted by the nonterminal S in this specification) to set apart the markup for greater readability. Such white space is typically not intended for inclusion in the delivered version of the document. On the other hand, "significant" white space that should be preserved in the delivered version is common, for example in poetry and source code.

An XML processor must always pass all characters in a document that are not markup through to the application. A validating XML processor must also inform the application which of these characters constitute white space appearing in element content.

A special attribute named `xml:space` may be attached to an element to signal an intention that in that element, white space should be preserved by applications. In valid documents, this attribute, like any other, must be declared if it is used. When declared, it must be given as an enumerated type whose only possible values are "default" and "preserve". For example:

```
<!ATTLIST poem   xml:space (default|preserve) 'preserve'>
```

The value "default" signals that applications' default white-space processing modes are acceptable for this element; the value "preserve" indicates the intent that applications preserve all the white space. This declared intent is considered to apply to all elements within the content of the element where it is specified, unless overriden with another instance of the `xml:space` attribute.

The root element of any document is considered to have signaled no intentions as regards application space handling, unless it provides a value for this attribute or the attribute is declared with a default value.

R-205

2.11 End-of-Line Handling

XML parsed entities are often stored in computer files which, for editing convenience, are organized into lines. These lines are typically separated by some combination of the characters carriage-return (#xD) and line-feed (#xA).

To simplify the tasks of applications, wherever an external parsed entity or the literal entity value of an internal parsed entity contains either the literal two-character sequence "#xD#xA" or a standalone literal #xD, an XML processor must pass to the application the single character #xA. (This behavior can conveniently be produced by normalizing all line breaks to #xA on input, before parsing.)

2.12 Language Identification

In document processing, it is often useful to identify the natural or formal language in which the content is written. A special attribute named `xml:lang` may be inserted in documents to specify the language used in the contents and attribute values of any element in an XML document. In valid documents, this attribute, like any other, must be declared if it is used. The values of the attribute are language identifiers as defined by [IETF RFC 1766], "Tags for the Identification of Languages":

Language Identification

```
[33] LanguageID ::= Langcode ('-' Subcode)*
[34]   Langcode ::= ISO639Code |  IanaCode |  UserCode
[35] ISO639Code ::= ([a-z] | [A-Z]) ([a-z] | [A-Z])
[36]   IanaCode ::= ('i' | 'I') '-' ([a-z] | [A-Z])+
[37]   UserCode ::= ('x' | 'X') '-' ([a-z] | [A-Z])+
[38]    Subcode ::= ([a-z] | [A-Z])+
```

The `Langcode` may be any of the following:

- a two-letter language code as defined by [ISO 639], "Codes for the representation of names of languages"
- a language identifier registered with the Internet Assigned Numbers Authority [IANA]; these begin with the prefix "`i-`" (or "`I-`")
- a language identifier assigned by the user, or agreed on between parties in private use; these must begin with the prefix "`x-`" or "`X-`" in order to ensure that they do not conflict with names later standardized or registered with IANA

There may be any number of `Subcode` segments; if the first subcode segment exists and the Subcode consists of two letters, then it must be a country code from [ISO 3166], "Codes for the representation of names of countries." If the first subcode consists of more than two letters, it must be a subcode for the language in question registered with IANA, unless the `Langcode` begins with the prefix "`x-`" or "`X-`".

It is customary to give the language code in lower case, and the country code (if any) in upper case. Note that these values, unlike other names in XML documents, are case insensitive.

For example:

```
<p xml:lang="en">The quick brown fox jumps over the lazy dog.</p>
<p xml:lang="en-GB">What colour is it?</p>
<p xml:lang="en-US">What color is it?</p>
<sp who="Faust" desc='leise' xml:lang="de">
  <l>Habe nun, ach! Philosophie,</l>
  <l>Juristerei, und Medizin</l>
  <l>und leider auch Theologie</l>
  <l>durchaus studiert mit heißem Bemüh'n.</l>
  </sp>
```

The intent declared with `xml:lang` is considered to apply to all attributes and content of the element where it is specified, unless overridden with an instance of `xml:lang` on another element within that content.

A simple declaration for `xml:lang` might take the form

```
xml:lang  NMTOKEN  #IMPLIED
```

but specific default values may also be given, if appropriate. In a collection of French poems for English students, with glosses and notes in English, the xml:lang attribute might be declared this way:

```
    <!ATTLIST poem   xml:lang NMTOKEN 'fr'>
    <!ATTLIST gloss  xml:lang NMTOKEN 'en'>
    <!ATTLIST note   xml:lang NMTOKEN 'en'>
```

3. Logical Structures

Each XML document contains one or more **elements**, the boundaries of which are either delimited by start-tags and end-tags, or, for empty elements, by an empty-element tag. Each element has a type, identified by name, sometimes called its "generic identifier" (GI), and may have a set of attribute specifications. Each attribute specification has a name and a value.

Element

[39] `element ::= EmptyElemTag`
`| STag content ETag [ WFC: Element Type Match ]`
`[ VC: Element Valid ]`

This specification does not constrain the semantics, use, or (beyond syntax) names of the element types and attributes, except that names beginning with a match to `(('X'|'x')('M'|'m')('L'|'l'))` are reserved for standardization in this or future versions of this specification.

Well-Formedness Constraint: Element Type Match
The `Name` in an element's end-tag must match the element type in the start-tag.

Validity Constraint: Element Valid
An element is valid if there is a declaration matching `elementdecl` where the `Name` matches the element type, and one of the following holds:

1. The declaration matches `EMPTY` and the element has no content.
2. The declaration matches `children` and the sequence of child elements belongs to the language generated by the regular expression in the content model, with optional white space (characters matching the nonterminal `S`) between each pair of child elements.

R-207

3. The declaration matches Mixed and the content consists of character data and child elements whose types match names in the content model.
4. The declaration matches ANY, and the types of any child elements have been declared.

3.1 Start-Tags, End-Tags, and Empty-Element Tags

The beginning of every non-empty XML element is marked by a **start-tag**.

Start-tag

```
[40]      STag ::= '<' Name (S Attribute)* S?   [ WFC: Unique Att Spec ]
                   '>'
[41] Attribute ::= Name Eq AttValue              [ VC: Attribute Value Type ]
                                                 [ WFC: No External Entity
                                                   References ]
                                                 [ WFC: No < in Attribute Values ]
```

The Name in the start- and end-tags gives the element's **type**. The Name-AttValue pairs are referred to as the **attribute specifications** of the element, with the Name in each pair referred to as the **attribute name** and the content of the AttValue (the text between the ' or " delimiters) as the **attribute value**.

Well-Formedness Constraint: Unique Att Spec
No attribute name may appear more than once in the same start-tag or empty-element tag.

Validity Constraint: Attribute Value Type
The attribute must have been declared; the value must be of the type declared for it. (For attribute types, see "3.3 Attribute-List Declarations".)

Well-Formedness Constraint: No External Entity References
Attribute values cannot contain direct or indirect entity references to external entities.

Well-Formedness Constraint: No < in Attribute Values
The replacement text of any entity referred to directly or indirectly in an attribute value (other than "<") must not contain a <.

An example of a start-tag:

```
<termdef id="dt-dog" term="dog">
```

The end of every element that begins with a start-tag must be marked by an **end-tag** containing a name that echoes the element's type as given in the start-tag:

End-tag

```
[42] ETag ::= '</' Name S? '>'
```

An example of an end-tag:

```
</termdef>
```

The text between the start-tag and end-tag is called the element's **content**:

R-208

Content of Elements

[43] content ::= (element | CharData | Reference | CDSect | PI | Comment)*

If an element is **empty**, it must be represented either by a start-tag immediately followed by an end-tag or by an empty-element tag. An **empty-element tag** takes a special form:

Tags for Empty Elements

[44] EmptyElemTag ::= '<' Name (S Attribute)* S? '/>' [WFC: Unique Att Spec]

Empty-element tags may be used for any element which has no content, whether or not it is declared using the keyword EMPTY. For interoperability, the empty-element tag must be used, and can only be used, for elements which are declared EMPTY.

Examples of empty elements:

```
<IMG align="left"
 src="http://www.w3.org/Icons/WWW/w3c_home" />
<br></br>
<br/>
```

3.2 Element Type Declarations

The element structure of an XML document may, for validation purposes, be constrained using element type and attribute-list declarations. An element type declaration constrains the element's content.

Element type declarations often constrain which element types can appear as children of the element. At user option, an XML processor may issue a warning when a declaration mentions an element type for which no declaration is provided, but this is not an error.

An **element type declaration** takes the form:

Element Type Declaration

[45] elementdecl ::= '<!ELEMENT' S Name S contentspec S? '>' [VC: Unique Element Type Declaration]

[46] contentspec ::= 'EMPTY' | 'ANY' | Mixed | children

where the Name gives the element type being declared.

Validity Constraint: Unique Element Type Declaration
No element type may be declared more than once.

Examples of element type declarations:

```
<!ELEMENT br EMPTY>
<!ELEMENT p (#PCDATA|emph)* >
<!ELEMENT %name.para; %content.para; >
<!ELEMENT container ANY>
```

R-209

3.2.1 Element Content

An element type has **element content** when elements of that type must contain only child elements (no character data), optionally separated by white space (characters matching the nonterminal S). In this case, the constraint includes a content model, a simple grammar governing the allowed types of the child elements and the order in which they are allowed to appear. The grammar is built on content particles (cps), which consist of names, choice lists of content particles, or sequence lists of content particles:

Element-content Models

```
[47] children ::= (choice | seq) ('?' | '*' | '+')?
[48]       cp ::= (Name | choice | seq) ('?' | '*' | '+')?
[49]   choice ::= '(' S? cp ( S? '|' S? cp )* S? ')'     [ VC: Proper Group/PE Nesting ]
[50]      seq ::= '(' S? cp ( S? ',' S? cp )* S? ')'     [ VC: Proper Group/PE Nesting ]
```

where each Name is the type of an element which may appear as a child. Any content particle in a choice list may appear in the element content at the location where the choice list appears in the grammar; content particles occurring in a sequence list must each appear in the element content in the order given in the list. The optional character following a name or list governs whether the element or the content particles in the list may occur one or more (+), zero or more (*), or zero or one times (?). The absence of such an operator means that the element or content particle must appear exactly once. This syntax and meaning are identical to those used in the productions in this specification.

The content of an element matches a content model if and only if it is possible to trace out a path through the content model, obeying the sequence, choice, and repetition operators and matching each element in the content against an element type in the content model. For compatibility, it is an error if an element in the document can match more than one occurrence of an element type in the content model. For more information, see "E. Deterministic Content Models".

Validity Constraint: Proper Group/PE Nesting
Parameter-entity replacement text must be properly nested with parenthetized groups. That is to say, if either of the opening or closing parentheses in a `choice`, `seq`, or `Mixed` construct is contained in the replacement text for a parameter entity, both must be contained in the same replacement text. For interoperability, if a parameter-entity reference appears in a `choice`, `seq`, or `Mixed` construct, its replacement text should not be empty, and neither the first nor last non-blank character of the replacement text should be a connector (| or ,).

Examples of element-content models:

```
<!ELEMENT spec (front, body, back?)>
<!ELEMENT div1 (head, (p | list | note)*, div2*)>
<!ELEMENT dictionary-body (%div.mix; | %dict.mix;)*>
```

3.2.2 Mixed Content

An element type has **mixed content** when elements of that type may contain character data, optionally interspersed with child elements. In this case, the types of the child elements may be constrained, but not their order or their number of occurrences:

Mixed-content Declaration

```
[51] Mixed ::=  '(' S? '#PCDATA' (S? '|' S? Name)*
                S? ')*'
                | '(' S? '#PCDATA' S? ')'              [ VC: Proper Group/PE Nesting
                                                         ]
                                                       [ VC: No Duplicate Types ]
```

where the Names give the types of elements that may appear as children.

Validity Constraint: No Duplicate Types
The same name must not appear more than once in a single mixed-content declaration.

Examples of mixed content declarations:

```
<!ELEMENT p (#PCDATA|a|ul|b|i|em)*>
<!ELEMENT p (#PCDATA | %font; | %phrase; | %special; | %form;)* >
<!ELEMENT b (#PCDATA)>
```

3.3 Attribute-List Declarations

Attributes are used to associate name-value pairs with elements. Attribute specifications may appear only within start-tags and empty-element tags; thus, the productions used to recognize them appear in "3.1 Start-Tags, End-Tags, and Empty-Element Tags". Attribute-list declarations may be used:

- To define the set of attributes pertaining to a given element type.
- To establish type constraints for these attributes.
- To provide default values for attributes.

Attribute-list declarations specify the name, data type, and default value (if any) of each attribute associated with a given element type:

Attribute-list Declaration

```
[52] AttlistDecl ::=  '<!ATTLIST' S Name AttDef* S? '>'
[53]      AttDef ::=  S Name S AttType S DefaultDecl
```

The Name in the AttlistDecl rule is the type of an element. At user option, an XML processor may issue a warning if attributes are declared for an element type not itself declared, but this is not an error. The Name in the AttDef rule is the name of the attribute.

When more than one AttlistDecl is provided for a given element type, the contents of all those provided are merged. When more than one definition is provided for the same attribute of a given element type, the first declaration is binding and later declarations are ignored. For interoperability, writers of DTDs may choose to provide at most one attribute-list declaration for a given element type, at most one attribute definition for a given attribute name, and at least one attribute definition in each attribute-list declaration. For interoperability, an XML processor may at user option issue a warning when more than one attribute-list declaration is provided for a given element type, or more than one attribute definition is provided for a given attribute, but this is not an error.

3.3.1 Attribute Types

XML attribute types are of three kinds: a string type, a set of tokenized types, and enumerated types. The

R-211

string type may take any literal string as a value; the tokenized types have varying lexical and semantic constraints, as noted:

Attribute Types

```
[54]       AttType ::= StringType | TokenizedType
                     | EnumeratedType
[55]    StringType ::= 'CDATA'
[56] TokenizedType ::= 'ID'                          [ VC: ID ]
                                                     [ VC: One ID per
                                                       Element Type ]
                                                     [ VC: ID Attribute
                                                       Default ]
                     | 'IDREF'                       [ VC: IDREF ]
                     | 'IDREFS'                      [ VC: IDREF ]
                     | 'ENTITY'                      [ VC: Entity Name ]
                     | 'ENTITIES'                    [ VC: Entity Name ]
                     | 'NMTOKEN'                     [ VC: Name Token ]
                     | 'NMTOKENS'                    [ VC: Name Token ]
```

Validity Constraint: ID
Values of type ID must match the Name production. A name must not appear more than once in an XML document as a value of this type; i.e., ID values must uniquely identify the elements which bear them.

Validity Constraint: One ID per Element Type
No element type may have more than one ID attribute specified.

Validity Constraint: ID Attribute Default
An ID attribute must have a declared default of #IMPLIED or #REQUIRED.

Validity Constraint: IDREF
Values of type IDREF must match the Name production, and values of type IDREFS must match Names; each Name must match the value of an ID attribute on some element in the XML document; i.e. IDREF values must match the value of some ID attribute.

Validity Constraint: Entity Name
Values of type ENTITY must match the Name production, values of type ENTITIES must match Names; each Name must match the name of an unparsed entity declared in the DTD.

Validity Constraint: Name Token
Values of type NMTOKEN must match the Nmtoken production; values of type NMTOKENS must match Nmtokens.

Enumerated attributes can take one of a list of values provided in the declaration. There are two kinds of enumerated types:

Enumerated Attribute Types

```
[57] EnumeratedType ::= NotationType | Enumeration
[58]   NotationType ::= 'NOTATION' S '(' S? Name (S? '|'   [ VC: Notation
                        S? Name)* S? ')'                    Attributes ]
[59]    Enumeration ::= '(' S? Nmtoken (S? '|' S?          [ VC: Enumeration ]
                        Nmtoken)* S? ')'
```

R-212

A NOTATION attribute identifies a notation, declared in the DTD with associated system and/or public identifiers, to be used in interpreting the element to which the attribute is attached.

Validity Constraint: Notation Attributes
Values of this type must match one of the notation names included in the declaration; all notation names in the declaration must be declared.

Validity Constraint: Enumeration
Values of this type must match one of the Nmtoken tokens in the declaration.

For interoperability, the same Nmtoken should not occur more than once in the enumerated attribute types of a single element type.

3.3.2 Attribute Defaults

An attribute declaration provides information on whether the attribute's presence is required, and if not, how an XML processor should react if a declared attribute is absent in a document.

Attribute Defaults

```
[60] DefaultDecl ::= '#REQUIRED' | '#IMPLIED'
                   | (('#FIXED' S)? AttValue) [ VC: Required Attribute ]
                                              [ VC: Attribute Default Legal ]
                                              [ WFC: No < in Attribute Values ]
                                              [ VC: Fixed Attribute Default ]
```

In an attribute declaration, #REQUIRED means that the attribute must always be provided, #IMPLIED that no default value is provided. If the declaration is neither #REQUIRED nor #IMPLIED, then the AttValue value contains the declared **default** value; the #FIXED keyword states that the attribute must always have the default value. If a default value is declared, when an XML processor encounters an omitted attribute, it is to behave as though the attribute were present with the declared default value.

Validity Constraint: Required Attribute
If the default declaration is the keyword #REQUIRED, then the attribute must be specified for all elements of the type in the attribute-list declaration.

Validity Constraint: Attribute Default Legal
The declared default value must meet the lexical constraints of the declared attribute type.

Validity Constraint: Fixed Attribute Default
If an attribute has a default value declared with the #FIXED keyword, instances of that attribute must match the default value.

Examples of attribute-list declarations:

```
<!ATTLIST termdef
          id      ID      #REQUIRED
          name    CDATA   #IMPLIED>
<!ATTLIST list
          type    (bullets|ordered|glossary)  "ordered">
<!ATTLIST form
          method  CDATA   #FIXED "POST">
```

3.3.3 Attribute-Value Normalization

R-213

Before the value of an attribute is passed to the application or checked for validity, the XML processor must normalize it as follows:

- a character reference is processed by appending the referenced character to the attribute value
- an entity reference is processed by recursively processing the replacement text of the entity
- a whitespace character (#x20, #xD, #xA, #x9) is processed by appending #x20 to the normalized value, except that only a single #x20 is appended for a "#xD#xA" sequence that is part of an external parsed entity or the literal entity value of an internal parsed entity
- other characters are processed by appending them to the normalized value

If the declared value is not CDATA, then the XML processor must further process the normalized attribute value by discarding any leading and trailing space (#x20) characters, and by replacing sequences of space (#x20) characters by a single space (#x20) character.

All attributes for which no declaration has been read should be treated by a non-validating parser as if declared CDATA.

3.4 Conditional Sections

Conditional sections are portions of the document type declaration external subset which are included in, or excluded from, the logical structure of the DTD based on the keyword which governs them.

Conditional Section

```
[61]    conditionalSect ::= includeSect | ignoreSect
[62]       includeSect ::= '<![' S? 'INCLUDE' S? '[' extSubsetDecl ']]>'
[63]       ignoreSect ::= '<![' S? 'IGNORE' S? '[' ignoreSectContents* ']]>'
[64] ignoreSectContents ::= Ignore ('<![' ignoreSectContents ']]>' Ignore)*
[65]            Ignore ::= Char* - (Char* ('<![' | ']]>') Char*)
```

Like the internal and external DTD subsets, a conditional section may contain one or more complete declarations, comments, processing instructions, or nested conditional sections, intermingled with white space.

If the keyword of the conditional section is INCLUDE, then the contents of the conditional section are part of the DTD. If the keyword of the conditional section is IGNORE, then the contents of the conditional section are not logically part of the DTD. Note that for reliable parsing, the contents of even ignored conditional sections must be read in order to detect nested conditional sections and ensure that the end of the outermost (ignored) conditional section is properly detected. If a conditional section with a keyword of INCLUDE occurs within a larger conditional section with a keyword of IGNORE, both the outer and the inner conditional sections are ignored.

If the keyword of the conditional section is a parameter-entity reference, the parameter entity must be replaced by its content before the processor decides whether to include or ignore the conditional section.

An example:

```
<!ENTITY % draft 'INCLUDE' >
<!ENTITY % final 'IGNORE' >

<![%draft;[
<!ELEMENT book (comments*, title, body, supplements?)>
]]>
<![%final;[
<!ELEMENT book (title, body, supplements?)>
]]>
```

4. Physical Structures

An XML document may consist of one or many storage units. These are called **entities**; they all have **content** and are all (except for the document entity, see below, and the external DTD subset) identified by **name**. Each XML document has one entity called the document entity, which serves as the starting point for the XML processor and may contain the whole document.

Entities may be either parsed or unparsed. A **parsed entity's** contents are referred to as its replacement text; this text is considered an integral part of the document.

An **unparsed entity** is a resource whose contents may or may not be text, and if text, may not be XML. Each unparsed entity has an associated notation, identified by name. Beyond a requirement that an XML processor make the identifiers for the entity and notation available to the application, XML places no constraints on the contents of unparsed entities.

Parsed entities are invoked by name using entity references; unparsed entities by name, given in the value of ENTITY or ENTITIES attributes.

General entities are entities for use within the document content. In this specification, general entities are sometimes referred to with the unqualified term *entity* when this leads to no ambiguity. Parameter entities are parsed entities for use within the DTD. These two types of entities use different forms of reference and are recognized in different contexts. Furthermore, they occupy different namespaces; a parameter entity and a general entity with the same name are two distinct entities.

4.1 Character and Entity References

A **character reference** refers to a specific character in the ISO/IEC 10646 character set, for example one not directly accessible from available input devices.

Character Reference

```
[66] CharRef ::= '&#' [0-9]+ ';'
                 | '&#x' [0-9a-fA-F]+ ';' [ WFC: Legal Character ]
```

Well-Formedness Constraint: Legal Character
Characters referred to using character references must match the production for Char.

If the character reference begins with "&#x", the digits and letters up to the terminating ; provide a hexadecimal representation of the character's code point in ISO/IEC 10646. If it begins just with "&#", the digits up to the terminating ; provide a decimal representation of the character's code point.

An **entity reference** refers to the content of a named entity. References to parsed general entities use ampersand (&) and semicolon (;) as delimiters. **Parameter-entity references** use percent-sign (%) and semicolon (;) as delimiters.

R-215

Entity Reference

```
[67]   Reference ::= EntityRef | CharRef
[68]   EntityRef ::= '&' Name ';'          [ WFC: Entity Declared ]
                                           [ VC: Entity Declared ]
                                           [ WFC: Parsed Entity ]
                                           [ WFC: No Recursion ]
[69] PEReference ::= '%' Name ';'          [ VC: Entity Declared ]
                                           [ WFC: No Recursion ]
                                           [ WFC: In DTD ]
```

Well-Formedness Constraint: Entity Declared
In a document without any DTD, a document with only an internal DTD subset which contains no parameter entity references, or a document with "`standalone='yes'`", the Name given in the entity reference must match that in an entity declaration, except that well-formed documents need not declare any of the following entities: `amp`, `lt`, `gt`, `apos`, `quot`. The declaration of a parameter entity must precede any reference to it. Similarly, the declaration of a general entity must precede any reference to it which appears in a default value in an attribute-list declaration. Note that if entities are declared in the external subset or in external parameter entities, a non-validating processor is not obligated to read and process their declarations; for such documents, the rule that an entity must be declared is a well-formedness constraint only if standalone='yes'.

Validity Constraint: Entity Declared
In a document with an external subset or external parameter entities with "`standalone='no'`", the Name given in the entity reference must match that in an entity declaration. For interoperability, valid documents should declare the entities `amp`, `lt`, `gt`, `apos`, `quot`, in the form specified in "4.6 Predefined Entities". The declaration of a parameter entity must precede any reference to it. Similarly, the declaration of a general entity must precede any reference to it which appears in a default value in an attribute-list declaration.

Well-Formedness Constraint: Parsed Entity
An entity reference must not contain the name of an unparsed entity. Unparsed entities may be referred to only in attribute values declared to be of type `ENTITY` or `ENTITIES`.

Well-Formedness Constraint: No Recursion
A parsed entity must not contain a recursive reference to itself, either directly or indirectly.

Well-Formedness Constraint: In DTD
Parameter-entity references may only appear in the DTD.

Examples of character and entity references:

```
Type <key>less-than</key> (&#x3C;) to save options.
This document was prepared on &docdate; and
is classified &security-level;.
```

Example of a parameter-entity reference:

```
<!-- declare the parameter entity "ISOLat2"... -->
<!ENTITY % ISOLat2
          SYSTEM "http://www.xml.com/iso/isolat2-xml.entities" >
<!-- ... now reference it. -->
%ISOLat2;
```

R-216

4.2 Entity Declarations

Entities are declared thus:

Entity Declaration

```
[70] EntityDecl ::= GEDecl | PEDecl
[71]     GEDecl ::= '<!ENTITY' S Name S EntityDef S? '>'
[72]     PEDecl ::= '<!ENTITY' S '%' S Name S PEDef S? '>'
[73]  EntityDef ::= EntityValue | (ExternalID NDataDecl?)
[74]      PEDef ::= EntityValue | ExternalID
```

The Name identifies the entity in an entity reference or, in the case of an unparsed entity, in the value of an ENTITY or ENTITIES attribute. If the same entity is declared more than once, the first declaration encountered is binding; at user option, an XML processor may issue a warning if entities are declared multiple times.

4.2.1 Internal Entities

If the entity definition is an EntityValue, the defined entity is called an **internal entity**. There is no separate physical storage object, and the content of the entity is given in the declaration. Note that some processing of entity and character references in the literal entity value may be required to produce the correct replacement text: see "4.5 Construction of Internal Entity Replacement Text".

An internal entity is a parsed entity.

Example of an internal entity declaration:

```
<!ENTITY Pub-Status "This is a pre-release of the
 specification.">
```

4.2.2 External Entities

If the entity is not internal, it is an **external entity**, declared as follows:

External Entity Declaration

```
[75] ExternalID ::= 'SYSTEM' S SystemLiteral
                  | 'PUBLIC' S PubidLiteral S
                  SystemLiteral
[76]  NDataDecl ::= S 'NDATA' S Name                    [ VC: Notation Declared
                                                        ]
```

If the NDataDecl is present, this is a general unparsed entity; otherwise it is a parsed entity.

Validity Constraint: Notation Declared
The Name must match the declared name of a notation.

The SystemLiteral is called the entity's **system identifier**. It is a URI, which may be used to retrieve the entity. Note that the hash mark (#) and fragment identifier frequently used with URIs are not, formally, part of the URI itself; an XML processor may signal an error if a fragment identifier is given

R-217

as part of a system identifier. Unless otherwise provided by information outside the scope of this specification (e.g. a special XML element type defined by a particular DTD, or a processing instruction defined by a particular application specification), relative URIs are relative to the location of the resource within which the entity declaration occurs. A URI might thus be relative to the document entity, to the entity containing the external DTD subset, or to some other external parameter entity.

An XML processor should handle a non-ASCII character in a URI by representing the character in UTF-8 as one or more bytes, and then escaping these bytes with the URI escaping mechanism (i.e., by converting each byte to %HH, where HH is the hexadecimal notation of the byte value).

In addition to a system identifier, an external identifier may include a **public identifier.** An XML processor attempting to retrieve the entity's content may use the public identifier to try to generate an alternative URI. If the processor is unable to do so, it must use the URI specified in the system literal. Before a match is attempted, all strings of white space in the public identifier must be normalized to single space characters (#x20), and leading and trailing white space must be removed.

Examples of external entity declarations:

```
<!ENTITY open-hatch
         SYSTEM "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY open-hatch
         PUBLIC "-//Textuality//TEXT Standard open-hatch boilerplate//EN"
         "http://www.textuality.com/boilerplate/OpenHatch.xml">
<!ENTITY hatch-pic
         SYSTEM "../grafix/OpenHatch.gif"
         NDATA gif >
```

4.3 Parsed Entities

4.3.1 The Text Declaration

External parsed entities may each begin with a **text declaration**.

Text Declaration
[77] `TextDecl ::= '<?xml' VersionInfo? EncodingDecl S? '?>'`

The text declaration must be provided literally, not by reference to a parsed entity. No text declaration may appear at any position other than the beginning of an external parsed entity.

4.3.2 Well-Formed Parsed Entities

The document entity is well-formed if it matches the production labeled `document`. An external general parsed entity is well-formed if it matches the production labeled `extParsedEnt`. An external parameter entity is well-formed if it matches the production labeled `extPE`.

Well-Formed External Parsed Entity
[78] `extParsedEnt ::= TextDecl? content`
[79] `extPE ::= TextDecl? extSubsetDecl`

An internal general parsed entity is well-formed if its replacement text matches the production labeled `content`. All internal parameter entities are well-formed by definition.

R-218

A consequence of well-formedness in entities is that the logical and physical structures in an XML document are properly nested; no start-tag, end-tag, empty-element tag, element, comment, processing instruction, character reference, or entity reference can begin in one entity and end in another.

4.3.3 Character Encoding in Entities

Each external parsed entity in an XML document may use a different encoding for its characters. All XML processors must be able to read entities in either UTF-8 or UTF-16.

Entities encoded in UTF-16 must begin with the Byte Order Mark described by ISO/IEC 10646 Annex E and Unicode Appendix B (the ZERO WIDTH NO-BREAK SPACE character, #xFEFF). This is an encoding signature, not part of either the markup or the character data of the XML document. XML processors must be able to use this character to differentiate between UTF-8 and UTF-16 encoded documents.

Although an XML processor is required to read only entities in the UTF-8 and UTF-16 encodings, it is recognized that other encodings are used around the world, and it may be desired for XML processors to read entities that use them. Parsed entities which are stored in an encoding other than UTF-8 or UTF-16 must begin with a text declaration containing an encoding declaration:

Encoding Declaration

```
[80] EncodingDecl ::= S 'encoding' Eq ('"' EncName '"' | "'" EncName "'" )
[81] EncName ::= [A-Za-z] ([A-Za-z0-9._] | '-')*   /* Encoding name contains only Latin characters */
```

In the document entity, the encoding declaration is part of the XML declaration. The `EncName` is the name of the encoding used.

In an encoding declaration, the values "`UTF-8`", "`UTF-16`", "`ISO-10646-UCS-2`", and "`ISO-10646-UCS-4`" should be used for the various encodings and transformations of Unicode / ISO/IEC 10646, the values "`ISO-8859-1`","`ISO-8859-2`", ... "`ISO-8859-9`" should be used for the parts of ISO 8859, and the values "`ISO-2022-JP`", "`Shift_JIS`", and "`EUC-JP`" should be used for the various encoded forms of JIS X-0208-1997. XML processors may recognize other encodings; it is recommended that character encodings registered (as *charsets*) with the Internet Assigned Numbers Authority [IANA], other than those just listed, should be referred to using their registered names. Note that these registered names are defined to be case-insensitive, so processors wishing to match against them should do so in a case-insensitive way.

In the absence of information provided by an external transport protocol (e.g. HTTP or MIME), it is an error for an entity including an encoding declaration to be presented to the XML processor in an encoding other than that named in the declaration, for an encoding declaration to occur other than at the beginning of an external entity, or for an entity which begins with neither a Byte Order Mark nor an encoding declaration to use an encoding other than UTF-8. Note that since ASCII is a subset of UTF-8, ordinary ASCII entities do not strictly need an encoding declaration.

It is a fatal error when an XML processor encounters an entity with an encoding that it is unable to process.

Examples of encoding declarations:

```
<?xml encoding='UTF-8'?>
<?xml encoding='EUC-JP'?>
```

4.4 XML Processor Treatment of Entities and References

The table below summarizes the contexts in which character references, entity references, and invocations of unparsed entities might appear and the required behavior of an XML processor in each case. The labels in the leftmost column describe the recognition context:

Reference in Content
as a reference anywhere after the start-tag and before the end-tag of an element; corresponds to the nonterminal `content`.

Reference in Attribute Value
as a reference within either the value of an attribute in a start-tag, or a default value in an attribute declaration; corresponds to the nonterminal `AttValue`.

Occurs as Attribute Value
as a `Name`, not a reference, appearing either as the value of an attribute which has been declared as type `ENTITY`, or as one of the space-separated tokens in the value of an attribute which has been declared as type `ENTITIES`.

Reference in Entity Value
as a reference within a parameter or internal entity's literal entity value in the entity's declaration; corresponds to the nonterminal `EntityValue`.

Reference in DTD
as a reference within either the internal or external subsets of the DTD, but outside of an `EntityValue` or `AttValue`.

	Entity Type				
	Parameter	Internal General	External Parsed General	Unparsed	C
Reference in Content	Not recognized	Included	Included if validating	Forbidden	In
Reference in Attribute Value	Not recognized	Included in literal	Forbidden	Forbidden	In
Occurs as Attribute Value	Not recognized	Forbidden	Forbidden	Notify	Not r
Reference in EntityValue	Included in literal	Bypassed	Bypassed	Forbidden	In
Reference in DTD	Included as PE	Forbidden	Forbidden	Forbidden	Fo

4.4.1 Not Recognized

Outside the DTD, the % character has no special significance; thus, what would be parameter entity references in the DTD are not recognized as markup in `content`. Similarly, the names of unparsed entities are not recognized except when they appear in the value of an appropriately declared attribute.

4.4.2 Included

An entity is **included** when its replacement text is retrieved and processed, in place of the reference itself, as though it were part of the document at the location the reference was recognized. The replacement text may contain both character data and (except for parameter entities) markup, which

R-220

must be recognized in the usual way, except that the replacement text of entities used to escape markup delimiters (the entities `amp`, `lt`, `gt`, `apos`, `quot`) is always treated as data. (The string "`AT&T;`" expands to "`AT&T;`" and the remaining ampersand is not recognized as an entity-reference delimiter.) A character reference is **included** when the indicated character is processed in place of the reference itself.

4.4.3 Included If Validating

When an XML processor recognizes a reference to a parsed entity, in order to validate the document, the processor must include its replacement text. If the entity is external, and the processor is not attempting to validate the XML document, the processor may, but need not, include the entity's replacement text. If a non-validating parser does not include the replacement text, it must inform the application that it recognized, but did not read, the entity.

This rule is based on the recognition that the automatic inclusion provided by the SGML and XML entity mechanism, primarily designed to support modularity in authoring, is not necessarily appropriate for other applications, in particular document browsing. Browsers, for example, when encountering an external parsed entity reference, might choose to provide a visual indication of the entity's presence and retrieve it for display only on demand.

4.4.4 Forbidden

The following are forbidden, and constitute fatal errors:

- the appearance of a reference to an unparsed entity.
- the appearance of any character or general-entity reference in the DTD except within an `EntityValue` or `AttValue`.
- a reference to an external entity in an attribute value.

4.4.5 Included in Literal

When an entity reference appears in an attribute value, or a parameter entity reference appears in a literal entity value, its replacement text is processed in place of the reference itself as though it were part of the document at the location the reference was recognized, except that a single or double quote character in the replacement text is always treated as a normal data character and will not terminate the literal. For example, this is well-formed:

```
<!ENTITY % YN '"Yes"' >
<!ENTITY WhatHeSaid "He said &YN;" >
```

while this is not:

```
<!ENTITY EndAttr "27'" >
<element attribute='a-&EndAttr;>
```

4.4.6 Notify

When the name of an unparsed entity appears as a token in the value of an attribute of declared type `ENTITY` or `ENTITIES`, a validating processor must inform the application of the system and public (if any) identifiers for both the entity and its associated notation.

4.4.7 Bypassed

When a general entity reference appears in the `EntityValue` in an entity declaration, it is bypassed and left as is.

R-221

4.4.8 Included as PE

Just as with external parsed entities, parameter entities need only be included if validating. When a parameter-entity reference is recognized in the DTD and included, its replacement text is enlarged by the attachment of one leading and one following space (#x20) character; the intent is to constrain the replacement text of parameter entities to contain an integral number of grammatical tokens in the DTD.

4.5 Construction of Internal Entity Replacement Text

In discussing the treatment of internal entities, it is useful to distinguish two forms of the entity's value. The **literal entity value** is the quoted string actually present in the entity declaration, corresponding to the non-terminal `EntityValue`. The **replacement text** is the content of the entity, after replacement of character references and parameter-entity references.

The literal entity value as given in an internal entity declaration (`EntityValue`) may contain character, parameter-entity, and general-entity references. Such references must be contained entirely within the literal entity value. The actual replacement text that is included as described above must contain the *replacement text* of any parameter entities referred to, and must contain the character referred to, in place of any character references in the literal entity value; however, general-entity references must be left as-is, unexpanded. For example, given the following declarations:

```
<!ENTITY % pub    "&#xc9;ditions Gallimard" >
<!ENTITY   rights "All rights reserved" >
<!ENTITY   book   "La Peste: Albert Camus,
&#xA9; 1947 %pub;. &rights;" >
```

then the replacement text for the entity "`book`" is:

```
La Peste: Albert Camus,
© 1947 Éditions Gallimard. &rights;
```

The general-entity reference "`&rights;`" would be expanded should the reference "`&book;`" appear in the document's content or an attribute value.

These simple rules may have complex interactions; for a detailed discussion of a difficult example, see "D. Expansion of Entity and Character References".

4.6 Predefined Entities

Entity and character references can both be used to **escape** the left angle bracket, ampersand, and other delimiters. A set of general entities (`amp`, `lt`, `gt`, `apos`, `quot`) is specified for this purpose. Numeric character references may also be used; they are expanded immediately when recognized and must be treated as character data, so the numeric character references "`&#60;`" and "`&#38;`" may be used to escape < and & when they occur in character data.

All XML processors must recognize these entities whether they are declared or not. For interoperability, valid XML documents should declare these entities, like any others, before using them. If the entities in question are declared, they must be declared as internal entities whose replacement text is the single character being escaped or a character reference to that character, as shown below.

```
<!ENTITY lt     "&#38;#60;">
<!ENTITY gt     "&#62;">
<!ENTITY amp    "&#38;#38;">
<!ENTITY apos   "&#39;">
<!ENTITY quot   "&#34;">
```

Note that the < and & characters in the declarations of "lt" and "amp" are doubly escaped to meet the requirement that entity replacement be well-formed.

4.7 Notation Declarations

Notations identify by name the format of unparsed entities, the format of elements which bear a notation attribute, or the application to which a processing instruction is addressed.

Notation declarations provide a name for the notation, for use in entity and attribute-list declarations and in attribute specifications, and an external identifier for the notation which may allow an XML processor or its client application to locate a helper application capable of processing data in the given notation.

Notation Declarations

[82] NotationDecl ::= '<!NOTATION' S Name S (ExternalID | PublicID) S? '>'

[83] PublicID ::= 'PUBLIC' S PubidLiteral

XML processors must provide applications with the name and external identifier(s) of any notation declared and referred to in an attribute value, attribute definition, or entity declaration. They may additionally resolve the external identifier into the system identifier, file name, or other information needed to allow the application to call a processor for data in the notation described. (It is not an error, however, for XML documents to declare and refer to notations for which notation-specific applications are not available on the system where the XML processor or application is running.)

4.8 Document Entity

The **document entity** serves as the root of the entity tree and a starting-point for an XML processor. This specification does not specify how the document entity is to be located by an XML processor; unlike other entities, the document entity has no name and might well appear on a processor input stream without any identification at all.

5. Conformance

5.1 Validating and Non-Validating Processors

Conforming XML processors fall into two classes: validating and non-validating.

Validating and non-validating processors alike must report violations of this specification's well-formedness constraints in the content of the document entity and any other parsed entities that they read.

Validating processors must report violations of the constraints expressed by the declarations in the DTD, and failures to fulfill the validity constraints given in this specification. To accomplish this, validating XML processors must read and process the entire DTD and all external parsed entities referenced in the document.

Non-validating processors are required to check only the document entity, including the entire internal

R-223

DTD subset, for well-formedness. While they are not required to check the document for validity, they are required to **process** all the declarations they read in the internal DTD subset and in any parameter entity that they read, up to the first reference to a parameter entity that they do *not* read; that is to say, they must use the information in those declarations to normalize attribute values, include the replacement text of internal entities, and supply default attribute values. They must not process entity declarations or attribute-list declarations encountered after a reference to a parameter entity that is not read, since the entity may have contained overriding declarations.

5.2 Using XML Processors

The behavior of a validating XML processor is highly predictable; it must read every piece of a document and report all well-formedness and validity violations. Less is required of a non-validating processor; it need not read any part of the document other than the document entity. This has two effects that may be important to users of XML processors:

- Certain well-formedness errors, specifically those that require reading external entities, may not be detected by a non-validating processor. Examples include the constraints entitled Entity Declared, Parsed Entity, and No Recursion, as well as some of the cases described as forbidden in "4.4 XML Processor Treatment of Entities and References".
- The information passed from the processor to the application may vary, depending on whether the processor reads parameter and external entities. For example, a non-validating processor may not normalize attribute values, include the replacement text of internal entities, or supply default attribute values, where doing so depends on having read declarations in external or parameter entities.

For maximum reliability in interoperating between different XML processors, applications which use non-validating processors should not rely on any behaviors not required of such processors. Applications which require facilities such as the use of default attributes or internal entities which are declared in external entities should use validating XML processors.

6. Notation

The formal grammar of XML is given in this specification using a simple Extended Backus-Naur Form (EBNF) notation. Each rule in the grammar defines one symbol, in the form

```
symbol ::= expression
```

Symbols are written with an initial capital letter if they are defined by a regular expression, or with an initial lower case letter otherwise. Literal strings are quoted.

Within the expression on the right-hand side of a rule, the following expressions are used to match strings of one or more characters:

`#xN`
where N is a hexadecimal integer, the expression matches the character in ISO/IEC 10646 whose canonical (UCS-4) code value, when interpreted as an unsigned binary number, has the value indicated. The number of leading zeros in the `#xN` form is insignificant; the number of leading zeros in the corresponding code value is governed by the character encoding in use and is not significant for XML.

`[a-zA-Z], [#xN-#xN]`
matches any character with a value in the range(s) indicated (inclusive).

`[^a-z], [^#xN-#xN]`
matches any character with a value *outside* the range indicated.

`[^abc], [^#xN#xN#xN]`
matches any character with a value not among the characters given.

R-224

`"string"`
matches a literal string matching that given inside the double quotes.

`'string'`
matches a literal string matching that given inside the single quotes.

These symbols may be combined to match more complex patterns as follows, where `A` and `B` represent simple expressions:

`(expression)`
`expression` is treated as a unit and may be combined as described in this list.

`A?`
matches `A` or nothing; optional `A`.

`A B`
matches `A` followed by `B`.

`A | B`
matches `A` or `B` but not both.

`A - B`
matches any string that matches `A` but does not match `B`.

`A+`
matches one or more occurrences of `A`.

`A*`
matches zero or more occurrences of `A`.

Other notations used in the productions are:

`/* ... */`
comment.

`[ wfc: ... ]`
well-formedness constraint; this identifies by name a constraint on well-formed documents associated with a production.

`[ vc: ... ]`
validity constraint; this identifies by name a constraint on valid documents associated with a production.

Appendices

A. References

A.1 Normative References

IANA
(Internet Assigned Numbers Authority) *Official Names for Character Sets*, ed. Keld Simonsen et al. See ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets.

IETF RFC 1766
IETF (Internet Engineering Task Force). *RFC 1766: Tags for the Identification of Languages*, ed. H. Alvestrand. 1995.

ISO 639
(International Organization for Standardization). *ISO 639:1988 (E). Code for the representation of names of languages.* [Geneva]: International Organization for Standardization, 1988.

ISO 3166
(International Organization for Standardization). *ISO 3166-1:1997 (E). Codes for the representation of names of countries and their subdivisions -- Part 1: Country codes* [Geneva]: International Organization for Standardization, 1997.

R-225

ISO/IEC 10646
ISO (International Organization for Standardization). *ISO/IEC 10646-1993 (E). Information technology -- Universal Multiple-Octet Coded Character Set (UCS) -- Part 1: Architecture and Basic Multilingual Plane.* [Geneva]: International Organization for Standardization, 1993 (plus amendments AM 1 through AM 7).
Unicode
The Unicode Consortium. *The Unicode Standard, Version 2.0.* Reading, Mass.: Addison-Wesley Developers Press, 1996.

A.2 Other References

Aho/Ullman
Aho, Alfred V., Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools.* Reading: Addison-Wesley, 1986, rpt. corr. 1988.
Berners-Lee et al.
Berners-Lee, T., R. Fielding, and L. Masinter. *Uniform Resource Identifiers (URI): Generic Syntax and Semantics.* 1997. (Work in progress; see updates to RFC1738.)
Brüggemann-Klein
Brüggemann-Klein, Anne. *Regular Expressions into Finite Automata.* Extended abstract in I. Simon, Hrsg., LATIN 1992, S. 97-98. Springer-Verlag, Berlin 1992. Full Version in Theoretical Computer Science 120: 197-213, 1993.
Brüggemann-Klein and Wood
Brüggemann-Klein, Anne, and Derick Wood. *Deterministic Regular Languages.* Universität Freiburg, Institut für Informatik, Bericht 38, Oktober 1991.
Clark
James Clark. Comparison of SGML and XML. See http://www.w3.org/TR/NOTE-sgml-xml-971215.
IETF RFC1738
IETF (Internet Engineering Task Force). *RFC 1738: Uniform Resource Locators (URL)*, ed. T. Berners-Lee, L. Masinter, M. McCahill. 1994.
IETF RFC1808
IETF (Internet Engineering Task Force). *RFC 1808: Relative Uniform Resource Locators*, ed. R. Fielding. 1995.
IETF RFC2141
IETF (Internet Engineering Task Force). *RFC 2141: URN Syntax*, ed. R. Moats. 1997.
ISO 8879
ISO (International Organization for Standardization). *ISO 8879:1986(E). Information processing -- Text and Office Systems -- Standard Generalized Markup Language (SGML).* First edition -- 1986-10-15. [Geneva]: International Organization for Standardization, 1986.
ISO/IEC 10744
ISO (International Organization for Standardization). *ISO/IEC 10744-1992 (E). Information technology -- Hypermedia/Time-based Structuring Language (HyTime).* [Geneva]: International Organization for Standardization, 1992. *Extended Facilities Annexe.* [Geneva]: International Organization for Standardization, 1996.

B. Character Classes

Following the characteristics defined in the Unicode standard, characters are classed as base characters (among others, these contain the alphabetic characters of the Latin alphabet, without diacritics), ideographic characters, and combining characters (among others, this class contains most diacritics); these classes combine to form the class of letters. Digits and extenders are also distinguished.

Characters
[84] Letter ::= BaseChar | Ideographic

R-226

[85] BaseChar ::= [#x0041-#x005A] | [#x0061-#x007A] | [#x00C0-#x00D6]
| [#x00D8-#x00F6] | [#x00F8-#x00FF] | [#x0100-#x0131]
| [#x0134-#x013E] | [#x0141-#x0148] | [#x014A-#x017E]
| [#x0180-#x01C3] | [#x01CD-#x01F0] | [#x01F4-#x01F5]
| [#x01FA-#x0217] | [#x0250-#x02A8] | [#x02BB-#x02C1]
| #x0386 | [#x0388-#x038A] | #x038C | [#x038E-#x03A1]
| [#x03A3-#x03CE] | [#x03D0-#x03D6] | #x03DA | #x03DC
| #x03DE | #x03E0 | [#x03E2-#x03F3] | [#x0401-#x040C]
| [#x040E-#x044F] | [#x0451-#x045C] | [#x045E-#x0481]
| [#x0490-#x04C4] | [#x04C7-#x04C8] | [#x04CB-#x04CC]
| [#x04D0-#x04EB] | [#x04EE-#x04F5] | [#x04F8-#x04F9]
| [#x0531-#x0556] | #x0559 | [#x0561-#x0586]
| [#x05D0-#x05EA] | [#x05F0-#x05F2] | [#x0621-#x063A]
| [#x0641-#x064A] | [#x0671-#x06B7] | [#x06BA-#x06BE]
| [#x06C0-#x06CE] | [#x06D0-#x06D3] | #x06D5
| [#x06E5-#x06E6] | [#x0905-#x0939] | #x093D
| [#x0958-#x0961] | [#x0985-#x098C] | [#x098F-#x0990]
| [#x0993-#x09A8] | [#x09AA-#x09B0] | #x09B2
| [#x09B6-#x09B9] | [#x09DC-#x09DD] | [#x09DF-#x09E1]
| [#x09F0-#x09F1] | [#x0A05-#x0A0A] | [#x0A0F-#x0A10]
| [#x0A13-#x0A28] | [#x0A2A-#x0A30] | [#x0A32-#x0A33]
| [#x0A35-#x0A36] | [#x0A38-#x0A39] | [#x0A59-#x0A5C]
| #x0A5E | [#x0A72-#x0A74] | [#x0A85-#x0A8B] | #x0A8D
| [#x0A8F-#x0A91] | [#x0A93-#x0AA8] | [#x0AAA-#x0AB0]
| [#x0AB2-#x0AB3] | [#x0AB5-#x0AB9] | #x0ABD | #x0AE0
| [#x0B05-#x0B0C] | [#x0B0F-#x0B10] | [#x0B13-#x0B28]
| [#x0B2A-#x0B30] | [#x0B32-#x0B33] | [#x0B36-#x0B39]
| #x0B3D | [#x0B5C-#x0B5D] | [#x0B5F-#x0B61]
| [#x0B85-#x0B8A] | [#x0B8E-#x0B90] | [#x0B92-#x0B95]
| [#x0B99-#x0B9A] | #x0B9C | [#x0B9E-#x0B9F]
| [#x0BA3-#x0BA4] | [#x0BA8-#x0BAA] | [#x0BAE-#x0BB5]
| [#x0BB7-#x0BB9] | [#x0C05-#x0C0C] | [#x0C0E-#x0C10]
| [#x0C12-#x0C28] | [#x0C2A-#x0C33] | [#x0C35-#x0C39]
| [#x0C60-#x0C61] | [#x0C85-#x0C8C] | [#x0C8E-#x0C90]
| [#x0C92-#x0CA8] | [#x0CAA-#x0CB3] | [#x0CB5-#x0CB9]
| #x0CDE | [#x0CE0-#x0CE1] | [#x0D05-#x0D0C]
| [#x0D0E-#x0D10] | [#x0D12-#x0D28] | [#x0D2A-#x0D39]
| [#x0D60-#x0D61] | [#x0E01-#x0E2E] | #x0E30
| [#x0E32-#x0E33] | [#x0E40-#x0E45] | [#x0E81-#x0E82]
| #x0E84 | [#x0E87-#x0E88] | #x0E8A | #x0E8D
| [#x0E94-#x0E97] | [#x0E99-#x0E9F] | [#x0EA1-#x0EA3]
| #x0EA5 | #x0EA7 | [#x0EAA-#x0EAB] | [#x0EAD-#x0EAE]
| #x0EB0 | [#x0EB2-#x0EB3] | #x0EBD | [#x0EC0-#x0EC4]
| [#x0F40-#x0F47] | [#x0F49-#x0F69] | [#x10A0-#x10C5]
| [#x10D0-#x10F6] | #x1100 | [#x1102-#x1103]
| [#x1105-#x1107] | #x1109 | [#x110B-#x110C]
| [#x110E-#x1112] | #x113C | #x113E | #x1140 | #x114C
| #x114E | #x1150 | [#x1154-#x1155] | #x1159
| [#x115F-#x1161] | #x1163 | #x1165 | #x1167 | #x1169
| [#x116D-#x116E] | [#x1172-#x1173] | #x1175 | #x119E
| #x11A8 | #x11AB | [#x11AE-#x11AF] | [#x11B7-#x11B8]
| #x11BA | [#x11BC-#x11C2] | #x11EB | #x11F0 | #x11F9
| [#x1E00-#x1E9B] | [#x1EA0-#x1EF9] | [#x1F00-#x1F15]
| [#x1F18-#x1F1D] | [#x1F20-#x1F45] | [#x1F48-#x1F4D]
| [#x1F50-#x1F57] | #x1F59 | #x1F5B | #x1F5D
| [#x1F5F-#x1F7D] | [#x1F80-#x1FB4] | [#x1FB6-#x1FBC]
| #x1FBE | [#x1FC2-#x1FC4] | [#x1FC6-#x1FCC]
| [#x1FD0-#x1FD3] | [#x1FD6-#x1FDB] | [#x1FE0-#x1FEC]
| [#x1FF2-#x1FF4] | [#x1FF6-#x1FFC] | #x2126
| [#x212A-#x212B] | #x212E | [#x2180-#x2182]
| [#x3041-#x3094] | [#x30A1-#x30FA] | [#x3105-#x312C]
| [#xAC00-#xD7A3]

[86] Ideographic ::= [#x4E00-#x9FA5] | #x3007 | [#x3021-#x3029]

[87] CombiningChar ::= | [#x0CCA-#x0CCD] | [#x0CD5-#x0CD6] | [#x0D02-#x0D03] | [#x

```
[88]          Digit ::= [#x0030-#x0039] | [#x0660-#x0669] | [#x06F0-#x06F9]
                      | [#x0966-#x096F] | [#x09E6-#x09EF] | [#x0A66-#x0A6F]
                      | [#x0AE6-#x0AEF] | [#x0B66-#x0B6F] | [#x0BE7-#x0BEF]
                      | [#x0C66-#x0C6F] | [#x0CE6-#x0CEF] | [#x0D66-#x0D6F]
                      | [#x0E50-#x0E59] | [#x0ED0-#x0ED9] | [#x0F20-#x0F29]
[89]       Extender ::= #x00B7 | #x02D0 | #x02D1 | #x0387 | #x0640 | #x0E46
                      | #x0EC6 | #x3005 | [#x3031-#x3035] | [#x309D-#x309E]
                      | [#x30FC-#x30FE]
```

The character classes defined here can be derived from the Unicode character database as follows:

- Name start characters must have one of the categories Ll, Lu, Lo, Lt, Nl.
- Name characters other than Name-start characters must have one of the categories Mc, Me, Mn, Lm, or Nd.
- Characters in the compatibility area (i.e. with character code greater than #xF900 and less than #xFFFE) are not allowed in XML names.
- Characters which have a font or compatibility decomposition (i.e. those with a "compatibility formatting tag" in field 5 of the database -- marked by field 5 beginning with a "<") are not allowed.
- The following characters are treated as name-start characters rather than name characters, because the property file classifies them as Alphabetic: [#x02BB-#x02C1], #x0559, #x06E5, #x06E6.
- Characters #x20DD-#x20E0 are excluded (in accordance with Unicode, section 5.14).
- Character #x00B7 is classified as an extender, because the property list so identifies it.
- Character #x0387 is added as a name character, because #x00B7 is its canonical equivalent.
- Characters ':' and '_' are allowed as name-start characters.
- Characters '-' and '.' are allowed as name characters.

C. XML and SGML (Non-Normative)

XML is designed to be a subset of SGML, in that every valid XML document should also be a conformant SGML document. For a detailed comparison of the additional restrictions that XML places on documents beyond those of SGML, see [Clark].

D. Expansion of Entity and Character References (Non-Normative)

This appendix contains some examples illustrating the sequence of entity- and character-reference recognition and expansion, as specified in "4.4 XML Processor Treatment of Entities and References".

If the DTD contains the declaration

```
<!ENTITY example "<p>An ampersand (&#38;#38;) may be escaped
numerically (&#38;#38;) or with a general entity
(&amp;amp;).</p>" >
```

then the XML processor will recognize the character references when it parses the entity declaration, and resolve them before storing the following string as the value of the entity "example":

```
<p>An ampersand (&#38;) may be escaped
numerically (&#38;#38;) or with a general entity
(&amp;amp;).</p>
```

A reference in the document to "&example;" will cause the text to be reparsed, at which time the start- and end-tags of the "p" element will be recognized and the three references will be recognized and

R-228

expanded, resulting in a "p" element with the following content (all data, no delimiters or markup):

```
An ampersand (&) may be escaped
numerically (&#38;) or with a general entity
(&amp;).
```

A more complex example will illustrate the rules and their effects fully. In the following example, the line numbers are solely for reference.

```
1 <?xml version='1.0'?>
2 <!DOCTYPE test [
3 <!ELEMENT test (#PCDATA) >
4 <!ENTITY % xx '&#37;zz;'>
5 <!ENTITY % zz '&#60;!ENTITY tricky "error-prone" >' >
6 %xx;
7 ]>
8 <test>This sample shows a &tricky; method.</test>
```

This produces the following:

- in line 4, the reference to character 37 is expanded immediately, and the parameter entity "xx" is stored in the symbol table with the value "%zz;". Since the replacement text is not rescanned, the reference to parameter entity "zz" is not recognized. (And it would be an error if it were, since "zz" is not yet declared.)
- in line 5, the character reference "<" is expanded immediately and the parameter entity "zz" is stored with the replacement text "<!ENTITY tricky "error-prone" >", which is a well-formed entity declaration.
- in line 6, the reference to "xx" is recognized, and the replacement text of "xx" (namely "%zz;") is parsed. The reference to "zz" is recognized in its turn, and its replacement text ("<!ENTITY tricky "error-prone" >") is parsed. The general entity "tricky" has now been declared, with the replacement text "error-prone".
- in line 8, the reference to the general entity "tricky" is recognized, and it is expanded, so the full content of the "test" element is the self-describing (and ungrammatical) string *This sample shows a error-prone method.*

E. Deterministic Content Models (Non-Normative)

For compatibility, it is required that content models in element type declarations be deterministic.

SGML requires deterministic content models (it calls them "unambiguous"); XML processors built using SGML systems may flag non-deterministic content models as errors.

For example, the content model ((b, c) | (b, d)) is non-deterministic, because given an initial b the parser cannot know which b in the model is being matched without looking ahead to see which element follows the b. In this case, the two references to b can be collapsed into a single reference, making the model read (b, (c | d)). An initial b now clearly matches only a single name in the content model. The parser doesn't need to look ahead to see what follows; either c or d would be accepted.

More formally: a finite state automaton may be constructed from the content model using the standard algorithms, e.g. algorithm 3.5 in section 3.9 of Aho, Sethi, and Ullman [Aho/Ullman]. In many such algorithms, a follow set is constructed for each position in the regular expression (i.e., each leaf node in the syntax tree for the regular expression); if any position has a follow set in which more than one following position is labeled with the same element type name, then the content model is in error and may be reported as an error.

Algorithms exist which allow many but not all non-deterministic content models to be reduced automatically to equivalent deterministic models; see Brüggemann-Klein 1991 [Brüggemann-Klein].

F. Autodetection of Character Encodings (Non-Normative)

The XML encoding declaration functions as an internal label on each entity, indicating which character encoding is in use. Before an XML processor can read the internal label, however, it apparently has to know what character encoding is in use--which is what the internal label is trying to indicate. In the general case, this is a hopeless situation. It is not entirely hopeless in XML, however, because XML limits the general case in two ways: each implementation is assumed to support only a finite set of character encodings, and the XML encoding declaration is restricted in position and content in order to make it feasible to autodetect the character encoding in use in each entity in normal cases. Also, in many cases other sources of information are available in addition to the XML data stream itself. Two cases may be distinguished, depending on whether the XML entity is presented to the processor without, or with, any accompanying (external) information. We consider the first case first.

Because each XML entity not in UTF-8 or UTF-16 format *must* begin with an XML encoding declaration, in which the first characters must be '`<?xml`', any conforming processor can detect, after two to four octets of input, which of the following cases apply. In reading this list, it may help to know that in UCS-4, '<' is "`#x0000003C`" and '?' is "`#x0000003F`", and the Byte Order Mark required of UTF-16 data streams is "`#xFEFF`".

- `00 00 00 3C`: UCS-4, big-endian machine (1234 order)
- `3C 00 00 00`: UCS-4, little-endian machine (4321 order)
- `00 00 3C 00`: UCS-4, unusual octet order (2143)
- `00 3C 00 00`: UCS-4, unusual octet order (3412)
- `FE FF`: UTF-16, big-endian
- `FF FE`: UTF-16, little-endian
- `00 3C 00 3F`: UTF-16, big-endian, no Byte Order Mark (and thus, strictly speaking, in error)
- `3C 00 3F 00`: UTF-16, little-endian, no Byte Order Mark (and thus, strictly speaking, in error)
- `3C 3F 78 6D`: UTF-8, ISO 646, ASCII, some part of ISO 8859, Shift-JIS, EUC, or any other 7-bit, 8-bit, or mixed-width encoding which ensures that the characters of ASCII have their normal positions, width, and values; the actual encoding declaration must be read to detect which of these applies, but since all of these encodings use the same bit patterns for the ASCII characters, the encoding declaration itself may be read reliably
- `4C 6F A7 94`: EBCDIC (in some flavor; the full encoding declaration must be read to tell which code page is in use)
- other: UTF-8 without an encoding declaration, or else the data stream is corrupt, fragmentary, or enclosed in a wrapper of some kind

This level of autodetection is enough to read the XML encoding declaration and parse the character-encoding identifier, which is still necessary to distinguish the individual members of each family of encodings (e.g. to tell UTF-8 from 8859, and the parts of 8859 from each other, or to distinguish the specific EBCDIC code page in use, and so on).

Because the contents of the encoding declaration are restricted to ASCII characters, a processor can reliably read the entire encoding declaration as soon as it has detected which family of encodings is in use. Since in practice, all widely used character encodings fall into one of the categories above, the XML encoding declaration allows reasonably reliable in-band labeling of character encodings, even when external sources of information at the operating-system or transport-protocol level are unreliable.

Once the processor has detected the character encoding in use, it can act appropriately, whether by invoking a separate input routine for each case, or by calling the proper conversion function on each character of input.

Like any self-labeling system, the XML encoding declaration will not work if any software changes the

R-230

entity's character set or encoding without updating the encoding declaration. Implementors of character-encoding routines should be careful to ensure the accuracy of the internal and external information used to label the entity.

The second possible case occurs when the XML entity is accompanied by encoding information, as in some file systems and some network protocols. When multiple sources of information are available, their relative priority and the preferred method of handling conflict should be specified as part of the higher-level protocol used to deliver XML. Rules for the relative priority of the internal label and the MIME-type label in an external header, for example, should be part of the RFC document defining the text/xml and application/xml MIME types. In the interests of interoperability, however, the following rules are recommended.

- If an XML entity is in a file, the Byte-Order Mark and encoding-declaration PI are used (if present) to determine the character encoding. All other heuristics and sources of information are solely for error recovery.
- If an XML entity is delivered with a MIME type of text/xml, then the `charset` parameter on the MIME type determines the character encoding method; all other heuristics and sources of information are solely for error recovery.
- If an XML entity is delivered with a MIME type of application/xml, then the Byte-Order Mark and encoding-declaration PI are used (if present) to determine the character encoding. All other heuristics and sources of information are solely for error recovery.

These rules apply only in the absence of protocol-level documentation; in particular, when the MIME types text/xml and application/xml are defined, the recommendations of the relevant RFC will supersede these rules.

G. W3C XML Working Group (Non-Normative)

This specification was prepared and approved for publication by the W3C XML Working Group (WG). WG approval of this specification does not necessarily imply that all WG members voted for its approval. The current and former members of the XML WG are:

Jon Bosak, Sun (Chair); James Clark (Technical Lead); Tim Bray, Textuality and Netscape (XML Co-editor); Jean Paoli, Microsoft (XML Co-editor); C. M. Sperberg-McQueen, U. of Ill. (XML Co-editor); Dan Connolly, W3C (W3C Liaison); Paula Angerstein, Texcel; Steve DeRose, INSO; Dave Hollander, HP; Eliot Kimber, ISOGEN; Eve Maler, ArborText; Tom Magliery, NCSA; Murray Maloney, Muzmo and Grif; Makoto Murata, Fuji Xerox Information Systems; Joel Nava, Adobe; Conleth O'Connell, Vignette; Peter Sharpe, SoftQuad; John Tigue, DataChannel